UCLouvain

École polytechnique de Louvain

Large-scale data visualization with BH t-SNE, LargeViz, and Umap

Author: **Di Wu**
Supervisors: **John Lee, Cyril De Bodt**
Reader: **Michel Verleysen**
Academic year 2019–2020
Master [120] in Data Sciences Engineering

Acknowledgements

This thesis has been completed with the help of many people that I would like to sincerely thank.

First, I would like to thank my thesis supervisor, Professor **John Lee** and his Phd student, Mr **Cyril De Bodt**, for the supervision of this thesis and its progress, for answering to my questions and giving me directions to be able to finalize the work. I thank Professor **Michel Verleysen** for accepting to read this thesis.

I would also like to thank the **Université Catholique de Louvain** for allowing me to study, learn, and finally make a thesis as future graduates, as well as all the higher members from the **Ecole Polytechnique de Louvain** who have guided me and have been part of my student life.

Finally, I would like to thank **my family**, who have supported me throughout our Bachelor and Master degrees, and without whom I would not have been allowed to be students.

Contents

Part I

Introduction

Chapter 1

Introduction

With the rapid development of information technology, many fields need to process a large amount of high-dimensional data. The speed of new data generation and storage is increasing rapidly, which leads to larger dimensions of the data. Due to the fact that high-dimensional data contains a lot of redundant information and the correlation between data is hidden in high-dimensional space, also considering that raw data are often sparse as a consequence of the curse of dimensionality, which leads to dimension disasters. As the number of features increases, the performance of the classifier will increase until the optimal number of features is reached. However, with more features added until dimension disaster happens, it will reduce the performance of the machine learning algorithm, which is also called Hughes phenomenon.

At present, data dimension reduction(DR) has become an important method for data mining, computer vision, machine learning, and pattern recognition to solve Hughes phenomenon and dimension disasters. The data DR method is to convert the data in the original high dimension(HD) space into a low dimension(LD) subspace, and reveals the essential distribution structure or pattern relationship of the data in the high-dimensional space. This not only reduces the time complexity of data processing and makes it easier to find data structure information, but also makes low-dimensional data representation easier to visualize.

The DR algorithms can be divided into two branches: Algorithms such as principal component analysis (PCA)[7] and Multidimensional scaling (MDS)[10] seek to preserve the distance structure within the data whereas algorithms like t-SNE[9], Isomap[12], LargeVis[2], Laplacian Eigenmaps[15] and Uniform manifold approximation and projection(UMAP)[13] favor the preservation of local distances over global distance.

This thesis mainly compares three DR algorithms: Barnes-Hut(BH) t-SNE[6], LargeVis and UMAP. They are all base on Stochastic Neighbor Embedding (SNE)[11] algorithm. While they also has differences on handling HD data, generating LD results as well as time consumption.

This work searches the characteristics of the algorithms in real data sets of different instances and dimensions, and uses neighborhood-based DR performance criteria to provide method performance for evaluating the performance of the results[1]. The work also generates heatmap for parameter pairs of each algorithm. By doing this, it discovers how each parameter changes the result and the similarities, as well as the differences between similar parameters in different algorithms and the reasons for the differences in performance between algorithms in the same data set. In the end, the optimal parameters of the three algorithms applied on each dataset with the help of the grid search algorithm. The examining code generates the visualization result with related DR performance score in 2D on each dataset and related time consumption. The applicability of each algorithm in different situations is also summarized.

The content is organized as follows: Part II first reviews three algorithms and the DR performance criteria used in this study. Part III details the experimental comparison results and draws the concludes.

Part II

Algorithms

Chapter 2

T-distributed Stochastic Neighbor Embedding (t-SNE)

Stochastic Neighbor Embedding(SNE) is a machine learning algorithm for visualization. It is a nonlinear dimensionality reduction technique that suits for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. Specifically, it maps similar data points in the high-dimensional space to the low-dimensional space so that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability in order to keep the relationship between data points. The conventional method is to use Euclidean distance to express this similarity, and SNE converts this distance relationship into a conditional probability to express the similarity.

2.1 Asymmetric SNE

Given N observations of some high dimensional data, for any pair, x_i and x_j, SNE defines the similarity (aka an affinity or weight) between them, using a Gaussian kernel function:

$$v_{j|i} = \exp\left(-\beta_i r_{ij}^2\right)$$

Where r_{ij} is the distance between x_i and x_j and β_i is determined by the method below. The notation of $v_{i|j}$ rather than v_{ij}, is to indicates that this quantity is not symmetric, i.e. $v_{i|j} \neq v_{j|i}$. This notation comes from the conditional versus joint probability definitions used in symmetric SNE (see below). The r_{ij} notation indicates that the distances are symmetric.

The weights are normalized to form N probability distributions:

$$p_{j|i} = \frac{\exp(-\mid\mid x_i - x_j \mid\mid^2 / 2\sigma_i^2)}{\sum_{i \neq k} \exp(-\mid\mid x_i - x_k \mid\mid^2 / 2\sigma_i^2}$$

Where σ_i represents the variance of the Gaussian distribution with x_i as the center point. When the data is mapped to the low-dimensional space, the similarity between high-dimensional data points should also be reflected in the data points in the low-dimensional space. If we assume that the mapping points of high-dimensional data points x_i and x_j in low-dimensional space are respectively y_i and y_j. At the same time, set the variance of all Gaussian distributions to $1/\sqrt{2}$, the conditional probability in low-dimensional space is represented by $q_{j|i}$:

$$q_{j|i} = \frac{\exp(-\mid\mid x_i - x_j \mid\mid^2 / 2\sigma_i^2)}{\sum_{i \neq k} \exp(-\mid\mid x_i - x_k \mid\mid^2 / 2\sigma_i^2}$$

As for symbol β_i, it is set by the value that results in the probability distribution having a specific perplexity. The perplexity has to be chosen by the user, but is interpreted as being a continuous version of the number of nearest neighbors, and generally is chosen to take values between 5 and 50. $p_{j|i}$ is a conditional probability, and is interpreted as the probability that item j will be chosen as being similar to item i, given that item i was picked already. At the same time, the output space of the embedded coordinates, which is the similarity between the points y_i and y_j is also defined as a Gaussian:

$$w_{ij} = \exp\left(-d_{ij}^2\right)$$

d_{ij} is the Euclidean distance between y_i and y_j as the mapping points of high-dimensional data points x_i and x_j in low-dimensional space, respectively. There is no β in this weight definition so these weights are symmetric. The output probabilities, $q_{j|i}$ are calculated from w_{ij} in the same way that we go from $v_{j|i}$ to $p_{j|i}$, again creating N probability distributions. Due to normalizing by rows, the $q_{j|i}$ are asymmetric despite the symmetric weights they are generated from.

Therefore, the SNE cost function could be defined as the sum of the Kullback-Leibler divergences of the N distributions:

$$C_{SNE} = \sum_i KL(P_i \mid\mid Q_i) = \sum_i^N \sum_j^N p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

The gradient of y_i of the SNE cost function is as follows:

$$\frac{\partial C_{SNE}}{\partial y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j)$$

As the KL distance is an asymmetric scale, the purpose of minimizing the cost function is to make the values of $p_{j|i}$ and $q_{j|i}$ as close as possible, which means the similarity of points in the low-dimensional space is consistent with the similarity of points in the high-dimensional space. But it can be trimmed from the form of the cost function. When $p_{j|i}$ is relatively bigger and $q_{j|i}$ is relatively smaller, the cost is higher; when $p_{j|i}$ is smaller and $q_{j|i}$ is bigger, the cost is lower. This means that, when two data points in a high-dimensional space are relatively close, if they are mapped to a low-dimensional space and are farther apart, then they will get a high penalty. In contrast, when the two data points in the high-dimensional space are farther apart, if they are mapped to the low-dimensional space, they will get a very low penalty value. This is a problem because there should be a higher penalty. In other words, the cost function of SNE pays more attention to the local structure rather than to the global impact.

2.2 Symmetric SNE

In Symmetric SNE, the input probability matrix is symmetrized by averaging $p_{j|i}$ and $p_{i|j}$ and then re-normalized over all pairs of points, to create a single (joint) probability distribution, p_{ij}:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}$$

The output probabilities, q_{ij} are now defined by normalizing the output weights over all pairs, again creating a single probability distribution:

$$q_{ij} = \frac{w_{ij}}{\sum_k^N \sum_l^N w_{kl}}$$

This definition not only satisfies the symmetry, but also ensures that the penalty value of x_i will not be too small[23]. Which means that no matter where the outlier's mapping point in low dimension y_i in the space is, the penalty value can be guaranteed[11]. At this time, the following cost function can be written using KL distance:

$$C_{SSNE} = \sum_i KL(P \mid\mid Q) = \sum_i^N \sum_j^N p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

The gradient turn out to be:

$$\frac{\partial C_{SSNE}}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

2.3 The Crowding Problem

By using the SNE method, the effect of the DR algorithm can be guaranteed, which means that the points of different clusters are more concentrated with each other. However, the boundary between each cluster is not always clear enough. It would be hard to tell the difference if there are no marks in different color for each group.

Part of the reason this happens, is because the SNE algorithm pays more attention to the local structure than the global structure. The most important reason could be the difference between the high-dimensional space distance distribution and the low-dimensional space distance distribution. With the increase of the dimensions, the sparseness of the high-dimensional spatial data will also increase because the volume increases exponentially.

If there is an m-dimensional sphere with radius r centered on data point x_i, its space increases with r^m. Assuming that the data points are uniformly distributed in the m-dimensional sphere, the distance between other data points and x_i as the dimension increases could be observed as below:

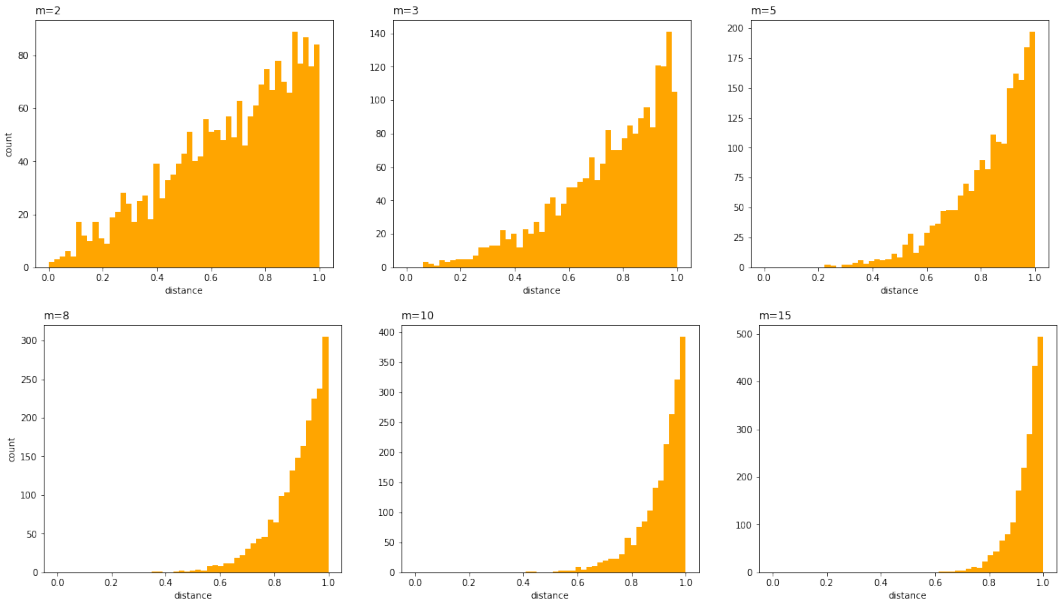


Figure 2.1: distribution of distances between x_i with different dimensions

It can be observed from the figure that as the dimension increases, most of the data points are clustered near the surface of the sphere, and the distance distribution from the point x_i is extremely uneven. If this distance relationship is directly retained to a low dimension, there will be a crowding problem.

2.4 t-SNE

t-SNE mainly adds two improvements on the basis of SNE: one is to transform SNE into symmetric SNE, and the other is to replace the original Gaussian distribution with t distribution in the low-dimensional space, and the high-dimensional space remains unchanged.

To understand t distribution, we can draw a random sample with a capacity of N from the normal population. If the mean of the normal population is μ, the variance is σ^2. The mean of the random sample is $\bar{x}$, and the variance is $s^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2$, and the random variable t can be expressed as:

$$t = \frac{\bar{x} - \mu}{s/\sqrt{N}}$$

t satisfies the t distribution with n−1 degrees of freedom, written as, $t \sim t(n-1)$. The t-distribution is symmetric and bell-shaped, like the normal distribution, but has heavier tails, meaning that it is more prone to producing values that fall far from its mean, which has obvious advantages when dealing with small samples and abnormal points.

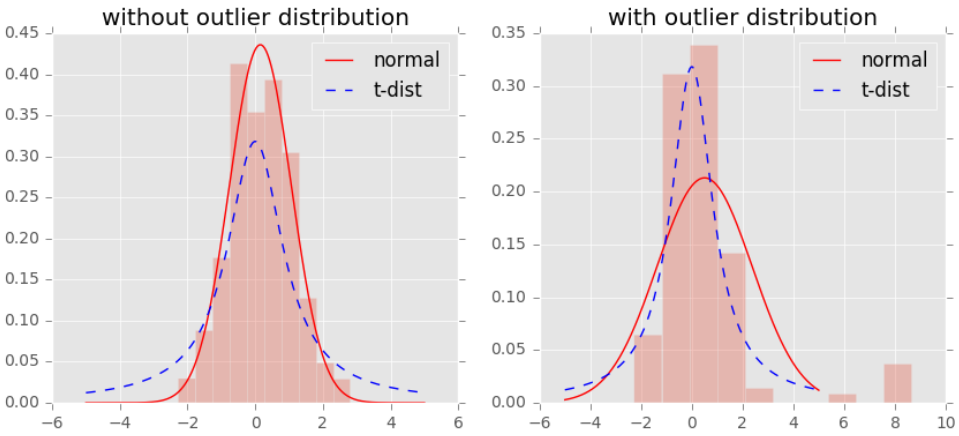


Figure 2.2: normal distribution and t distribution with and with out outliers

Figure 2.3 shows the difference when normal distribution and t-distribution handle outliers. It can be easily observed that when there are no outliers, the fitting result of t distribution and Gaussian distribution are basically consistent. In the second picture, there are some abnormal points. Because the tail of the Gaussian distribution is low, it is more sensitive to abnormal points. In order to take care of

these abnormal points, the fitting result of the Gaussian distribution deviates from the location of most samples, and the variance is also large. In contrast, the tail of the t distribution is relatively high and it is not sensitive to outliers, that ensure its robustness, so its fitting results are more reasonable, and the overall characteristics of the data are better captured. In order to avoid influence of outliers, we can then redefine the weight function from symmetric SNE with t-distribution:

$$w_{ij} = \frac{1}{1 + d_{ij}^2}$$

We then redefine q_{ij} using the t distribution with 1 degree of freedom as:

$$q_{ij} = \frac{(1+ \mid\mid y_i - y_j \mid\mid^2)^{-1}}{1 + d_{ij}^2}$$

The KL distance is still used to measure the similarity between the two distributions, and the gradient now is:

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1+ \mid\mid y_i - y_j \mid\mid^2)^{-1}$$

2.5 Barnes-Hut t-SNE(BH t-SNE)

BH t-SNE makes some improvements comparing to t-SNE algorithm with the help of tree-based algorithm, including two parts. The first one is the use of kNN graphs to represent the similarity of points in the high-dimensional space. The other one is the optimization of the gradient solution process. The gradient calculation is regarded as a joint force consisting of attraction and repulsion, and some optimization techniques are also used.

2.5.1 kNN graph for the similarity in HD space

In t-SNE, the overall distance distribution relationship in the high-dimensional space is expressed in the following form:

$$v_{j|i} = \exp\left(-\beta_i r_{ij}^2\right)$$

$$p_{j|i} = \frac{v_{j|i}}{\sum_k^N v_{k|i}}$$

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}$$

These three formulas show that every data point need to calculate the variance considering all other points. However, the probability p_{ij} that two distant points are neighbors to each other is very small and almost negligible. The efficiency of t-SNE is thus significantly reduced when processing large-scale high-dimensional data.

Therefore, when constructing a distance similarity relationship for a point in a high-dimensional space, it is not necessary to consider every node in the graph, but only a number of similar nodes. Here, we consider the $\lfloor 3k \rfloor$ closest points to the point x_i, where k is the perplexity of the conditional probability distribution around the point x_i. Only the set of these points are considered, and this will decrease the amount of calculation. The BH t-SNE algorithm uses a VP tree (vantage-point tree) to construct this kNN graph, and an accurate kNN graph can be obtained within $O(kN \log N)$ time complexity[6].

2.5.2 Approximation of repulsion in gradient

The t-SNE gradient could be regarded as a physical meaning, the gradient can be regarded as the joint force of all other points on y_i. If Z is set as $Z = \sum_{k \neq l}(1+ \mid y_k - y_l \mid^2)^{-1}$, then the gradient could be transformed by this feature:

$$\begin{aligned}
\frac{\partial C_{SSNE}}{\partial y_i} &= 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1+ \mid y_i - y_j \mid^2)^{-1} \\
&= 4 \sum_j (p_{ij} - q_{ij}) q_{ij} Z (y_i - y_j) \\
\\
\frac{\partial C_{SSNE}}{\partial y_i} &= 4(F_{attraction} + F_{repulsion}) \\
&= 4(\sum_{i \neq j} p_{ij} q_{ij} Z (y_i - y_j) - \sum_{i \neq j} q_{ij}^2 Z (y_i - y_j))
\end{aligned}$$

It can be seen that after decomposing the gradient into two parts of attraction and repulsion. The position of point y_i in the low-dimensional space is determined by the joint force of all other points on it. A certain point y_j put its force along the direction of $y_i - y_j$. Whether attraction or repulsion is dominant, it depends on the distance between y_j and y_i.

As for the formula above, the first part is easier to calculate since the $q_{ij} Z = 1+ \mid y_i - y_j \mid^2$ could use the approximation above, which only considers the nearest

neighbor nodes, while ignoring distant nodes with time complexity is $O(uN)$. However, there is still a large amount of calculation to calculate repulsion with time complexity $O(N^2)$ because the distribution of P and Q is different, there are still some optimization techniques to simplify this calculation.

If there are three data points (y_i, y_j, y_k) as shown below, their distance relationship can be summarized as $y_i - y_j \approx y_i - y_k >> y_j - y_k$, the repulsion of y_j and y_k to y_i is approximately equal.

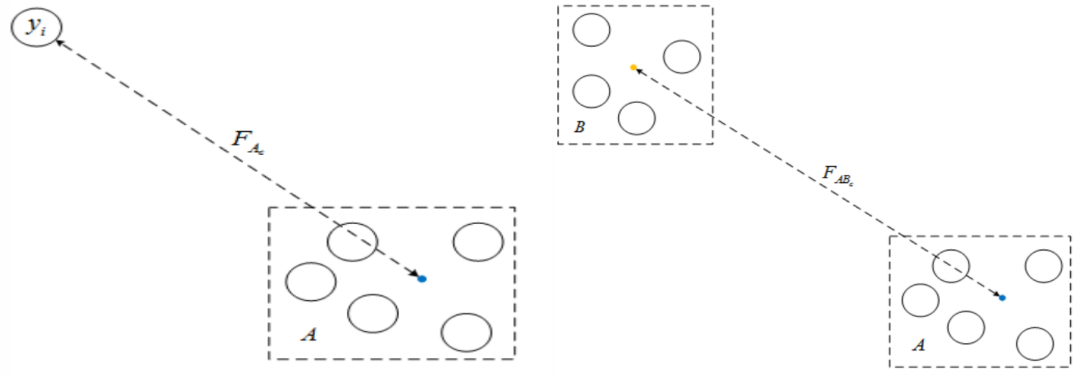
Figure 2.3: Repulsion between y_j and y_k to y_i

This situation is very common in low-dimensional space, and even the repulsion of each point in a certain area can be approximated by the same value. If a point is given arbitrarily, we just need to calculate the repulsion between given point and the centroid of the data points within the certain area, and use the optimization method just now to calculate the total repulsion for them.

The figure 2.4(a) shows the situation for point to area. If we assume that the repulsion generated by the five points in area A is approximately equal to y_i, then we can calculate the repulsion F_{A_c} generated by the center point of these five points, and the total repulsive force generated by area A is $5F_{A_c}$. The quad tree is used to complete the area search task[6], and uses the repulsion generated by the center of the area as the representative value of the entire area. Of course, not all areas meet this approximate condition. Here, the Barnes-Hut algorithm is used to search and verify that the approximate conditions are met[6].

2.5.3 From point-area to area-area

In the above content, we consider the approximation of the repulsive force between a point and an area. In fact, we can further optimize the approximation scheme of the repulsive force between one area and another area.



(a) point to area case (b) area to area case

Figure 2.4: Approximation of joint force calculation

The figure 2.4(b) shows further optimization, considering the approximation of the repulsion between one area and another. The repulsion generated by any two nodes between the two regions A and B can be approximated by F_{AB_c}. Similarly, it is also necessary to determine whether the repulsive force between the two regions meets the approximate conditions.

By constructing the relationship between the area and the area instead of points to points, the calculation of attraction and repulsion force of cost function reduced. This happens because we only considering the relationship between the limited areas, and the amount of calculation is significantly decreased from $O(n^2)$ to $O(nlogn)$[6]. Considering the results of t-SNE and BH t-SNE are similar[6] and latter one is much faster than another. This thesis will focus the performance and characteristics of BH t-SNE in the experiment part.

Chapter 3

LargeVis

Although t-SNE works well on small-scale data, there are still some disadvantages of t-SNE:

1. The efficiency of t-SNE is significantly reduced when processing large-scale high-dimensional data (including BH t-SNE).
2. from the experiments, we can observe that the parameters of t-SNE are sensitive to different data sets, which means that even finding out the proper parameters setting for t-SNE in one data set with a good visualization result, the parameters cannot be applied to another data set. We can also observe that LargeVis's parameter setting is much more stable and it also provide a better visualization performance with default values on average.

As can be seen from the name, the goal of LargeVis[2] is the visualization of large-scale data sets, which takes a different approach comparing with t-SNE: it reuses many of the same definitions as t-SNE but it is sufficiently modified so that stochastic gradient descent can be used. The main improvement is the efficient algorithm for constructing kNN graphs, as well as the low-dimensional space probability calculation formula and the objective function.

3.1 Efficient kNN graph constructing algorithm

In the BH t-SNE, for high-dimensional spatial distance similarity, it only considers several closest neighbor points to x_i, which is essentially a process of constructing a kNN graph. The VP(vantage-point) tree is used to construct an accurate kNN graph, but the efficiency is still not good, especially for the large-scale dataset. Moreover, LargeVis uses a way more ingenious. Indeed, instead of pursuing a one-step approach, it first approximates and then improves accuracy.

3.1.1 K-D tree and Random Projection Tree

There are generally three types of common methods for building KNN graph: the first type is the space-partitioning trees algorithm, the second type is the locality sensitive hashing algorithm, and the third type is the neighbor exploring techniques algorithm. Among them, k-d tree and random projection tree belong to the first type of algorithm.

A k-d tree is a data structure that partitions a k-dimensional data space, and is essentially a binary tree. It is mainly used for searching key data in multi-dimensional space, such as range search and nearest neighbor search. The following figure shows the use of k-d trees to search for k nearest neighbors in a two-dimensional space, and then construct a kNN graph. It constructs a kd tree through a recursive process. The root node corresponds to all points in the area. After dividing the space into a left subtree and a right subtree in a certain dimension, the next level of child nodes can be obtained by repeating the splitting process of the root node. Until the number of points corresponding to all leaf nodes in the kd tree is less than a certain threshold.

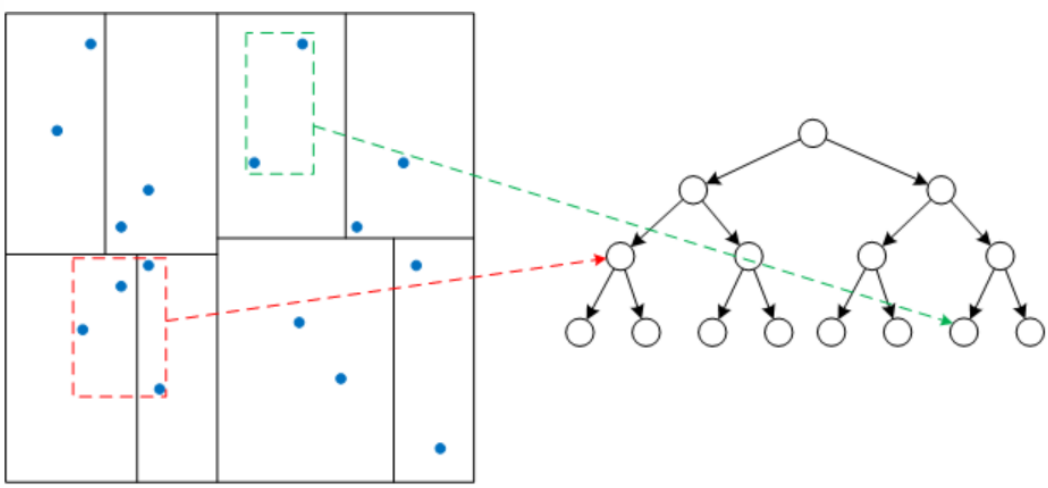


Figure 3.1: k-d tree for dimension 2

With the k-d tree, we don't have to calculate the distance between a certain point and all other points one by one to find k nearest neighbors. For example, to find the k-nearest neighbors of the red dot in the figure below, it only needs to search the current subspace, and at the same time, continue to backtrack and search other subspaces of the parent node to find the k-nearest neighbors.

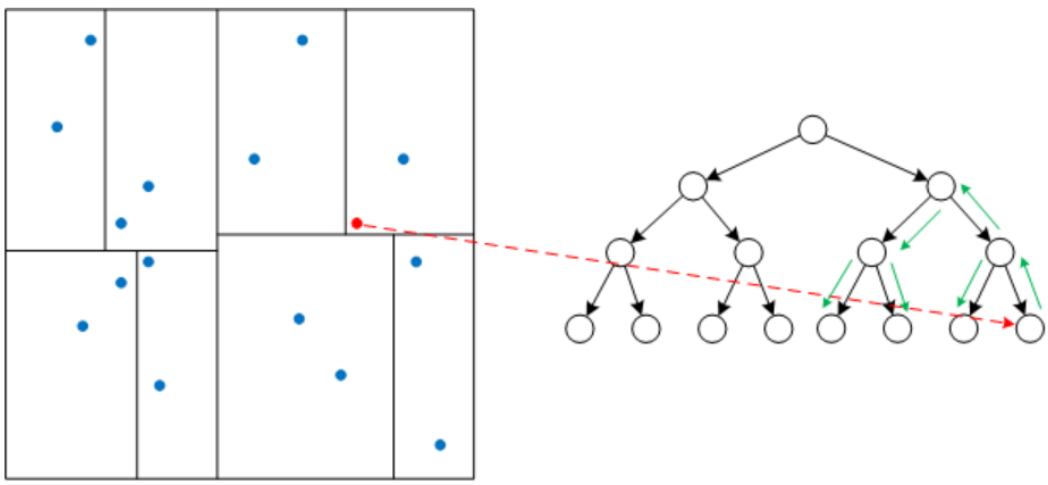


Figure 3.2: finding k-nearest neighbors for k-d tree

However, the biggest problem with the k-d tree is that the way it divides the space is relatively rigid, which is strictly based on the coordinate axis. For high-dimensional data, each dimension of the high-dimensional data is used as a coordinate axis. When the instances of data is high, the depth of the k-d tree could be a lot also, which may also lead to the dimensionality disaster problem.

In contrast, the way of dividing the space by random projection tree is more flexible. It divides all the data, reduces the number of points for each search and calculates to an acceptable range. Then it builds multiple random projection trees to form a random projection forest, and uses the comprehensive result of the forest as the final result. Although the basic idea of the random projection tree is similar to the k-d tree, but the way to divide the space is not according to the coordinate axis, while according to the randomly generated unit vector in order to find leaf nodes belong to it[18]. This is the process for calculating one random projection tree. Using the same method, multiple random projection trees are established to form a random forest, and the sum result of the forest is regarded as the final result. At the same time, because the data in principle is on a manifold, instead of a disorderly manner. Therefore, the depth of the random projection tree is not determined by the dimensionality of the data, but depends on the dimensionality of the manifold where the data is located[3].

When a random projection tree is used to find k nearest neighbors, the backtracking search method is similar to the k-d tree. If the requirement of the tree's accuracy is not high, the characteristics of the random projection tree can be used to adopt

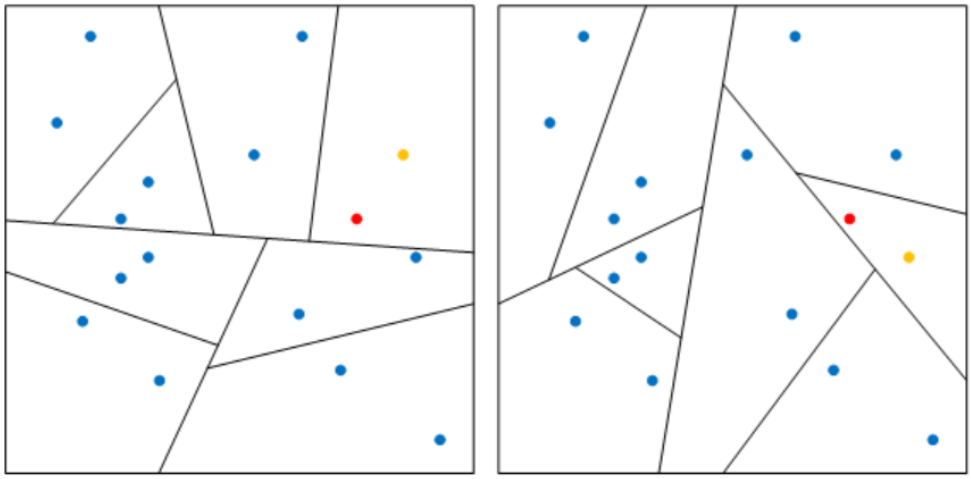


Figure 3.3: finding k-nearest neighbors for random projection tree

a simpler and time-saving method. In other words, we can construct multiple random projection trees in parallel. Since the divided unit vectors are generated randomly, each random projection tree divides the current space differently, as shown in the following figure with two-dimensional space as an example. If the k-nearest neighbors of the red dot need to be fixed, it only needs to search the subspace where it is located in different random projection trees, and finally take the union. Although this is time-consuming and space-consuming in the process of constructing a random projection tree, it is very efficient in the search phase.

3.1.2 The neighbor's neighbor may also be my neighbor

The idea of "neighbor's neighbor may also be my neighbor", start from an initial nearest-neighbor graph. The algorithm iteratively refines the graph by exploring the neighbors of neighbors defined according to the current graph[2]. It uses the neighbor search algorithm to find potential neighbors, calculating the distance between the neighbor and the current point, the neighbor's neighbor and the current point. With the k nodes with the smallest distance as k nearest neighbors, finally we will get an accurate kNN graph.

3.2 Low-dimensional space visualization algorithm

After completing the KNN graph algorithm for high dimensional space, we need to project the nodes of the graph into a low-dimensional space. In the process of low-dimensional space visualization, the idea of t-SNE is to ensure that the

distance distribution P of the high-dimensional space and the distance distribution Q of the low-dimensional space are as close as possible, and use the KL distance to write the cost function and find the gradient. However, the efficiency problem has not been solved well.

In LargeVis algorithm, in order to solve the efficiency problem, a probability model is applied to it in order to maintain the similarity of vertices in low-dimensional space. That is, the algorithm will bring similar vertices close to each other in a low-dimensional space, and separate different vertices from each other. It uses negative sampling optimization method from Word2vec to achieve the goal.

3.2.1 Negative sampling

Negative sampling in word2vec

Word2vec is a tool for calculating word vectors released by Google in 2013. It can measure the similarity between words using two kinds of architectures 1)Continuous Bag of Words (CBOW) and 2)Skip-gram. While negative sampling is the main optimization technique for word2vec.

Let's take the example of the Skip-gram model as an example
If we take the Skip-gram model as an example, its idea is to predict the context from the target word, and to use a context window in order to limit the text range. It needs to maximize the probability that the context window appears around the target word, which is to maximize $p(c \mid w) = \sum p(wi \mid w)$. The words $w_i \in c$ that appear in the context window around the target word constitute a positive sample (wi, w), and the words $w_j \in D$ that do not appear around the target word constitute a negative sample (w_j, w). The goal of this step is to maximize the probability of positive samples during the training process, while also reducing the probability of negative samples.

Due to the large number of negative samples (words outside the context window can basically constitute negative samples), it is obviously unrealistic to directly consider all negative samples, so we can select a part of negative samples by sampling. Because some words appear frequently in the corpus and some words appear low, it is obviously not advisable to draw negative samples directly from the vocabulary. Word2vec uses a weighted sampling strategy, that is, sampling according to word frequency. High-frequency words are more likely to be sampled, and low-frequency words are less likely to be sampled.

As we can see in figure 3.5, the upper line segment is divided by word frequency, the

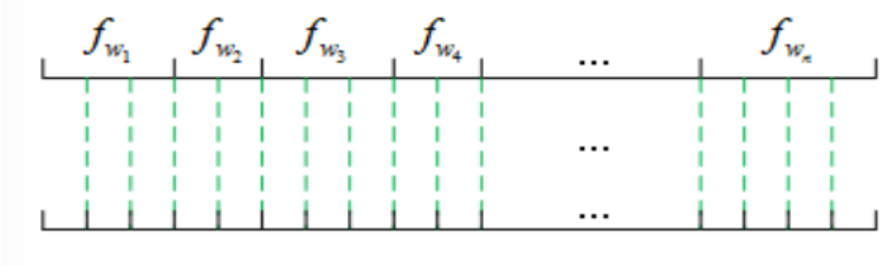


Figure 3.4: Weighted sampling details

higher the word frequency, the longer the line segment, and the lower line segment is divided equally. We randomly place dots in the lower line segment (uniformly distributed), and according to the interval where the point falls, correspond to the upper line segment to determine the word sampled. Intuitively, in this way, words with higher word frequency are more likely to be sampled, and words with lower word frequency are less likely to be sampled.After adding negative sampling optimization, the form of the objective function becomes:

$$log\sigma(v_{w_c}^T \cdot v_w) + \sum_{\substack{i=1 \\ E_{w_i} \sim P_n(f)}}^{k} log\sigma(-v_{w_c}^T \cdot v_w)$$

Where w represents the target word, w_c represents the word in the context window around the target word (positive sample), w_i represents the word that does not appear in the context window (negative sample), k represents the number of negative samples extracted, and $P_n(f)$ is Used for noise distribution generated by negative samples, f represents word frequency, $P_n(f) \propto f^{0.75}$. The exponent is suggested to set as 0.75[4], this makes the data closer to the center, which means that the higher values tend to lower and the lower values do the opposite in order to avoid high frequent words appearing all the time.

3.2.2 Negative sampling in network

It is actually similar to use negative sampling in the network instead of word2vec. We can regard the current center point as the target word, and its neighbor nodes as words appearing in the context window. Then the center point and its neighbor nodes constitute a positive sample, and the center point as well as the non-neighbor points constitute a negative sample.

As a small part of the kNN diagram shown in the figure 3.5(a) below, if the blue point is the center point, each green point and the blue point can form a positive sample, and each yellow point and the blue point form a negative sample. If we

use y_i and y_j to represent two points in a low-dimensional space, the probability that the two points have a binary edge $e_{ij} = 1$ (the edge with weight 1) in the kNN graph is:

$$P(e_{ij} = 1) = f(|| y_i - y_j ||^2)$$

Among them, $f(\cdot)$ similarly uses the t distribution in t-SNE, $f(x) = \frac{1}{1+x^2}$. If the distance between y_i and y_j is smaller, the probability of two points having a binary edge in the kNN graph is higher. On the contrary, if the distance between y_i and y_j is larger, the probability that two points have binary edges in the kNN graph is smaller. LargeVis also considers the weighted network, and defines the probability that the edge weight is w_{ij} as:

$$P(e_{ij} = w_{ij}) = P(e_{ij} = 1)^{w_{ij}}$$

If the positive sample is E, and the set of negative sample is $\overline{E}$. Both the set of positive samples and negative samples can be obtained directly through the kNN graph. The optimization goal can be written as:

$$O = \prod_{(i,j)\in E} p(e_{ij} = 1)^{w_{ij}} \prod_{(i,j)\in \overline{E}} (1 - p(e_{ij} = 1))^{\gamma}$$

The whole optimization goal can be understood as maximizing the probability of a node pair with a positive sample having a connected edge in the kNN graph, and minimizing the probability of a node pair with a negative sample having a connected edge in the kNN graph. Where is the weight we set uniformly for the negative sample edge. Here again take a logarithm to the formula, the optimization goal becomes:

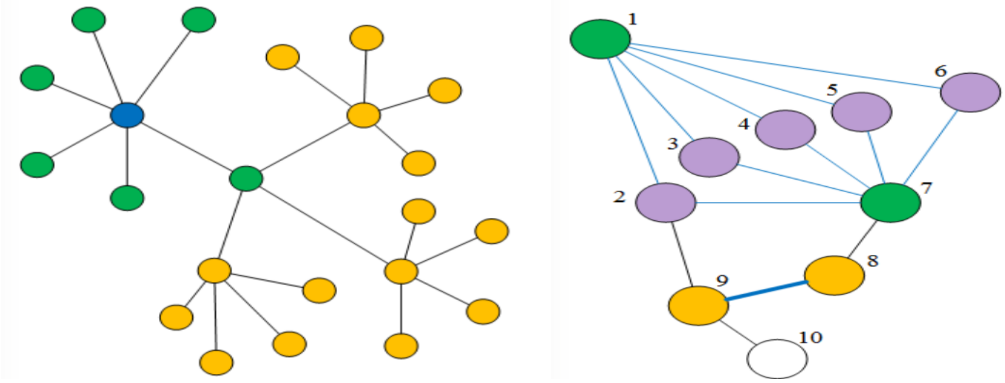
$$O = \sum_{(i,j)\in E} p(e_{ij} = 1)^{w_{ij}} \sum_{(i,j)\in \overline{E}} (1 - p(e_{ij} = 1))^{\gamma}$$

Here we are using all negative samples $\bar{E}$, the calculation is too large. With the help of negative sampling algorithm, for each point i, we randomly select M points and i to form a negative sample. The distribution of the points follows a noise distribution $P_n(j)$ with $P_n(j) \propto d_j^{0.75}$, where d_j is the degree of point j (here, the degree of the node is used, and word2vec is the word frequency of the word). We can redefine the objective function:

$$O = \sum_{(i,j)\in E} w_{ij}(p(e_{ij} = 1) + \sum_{\substack{k=1 \\ E_{i_k} \sim P_n(j)}}^{k} \gamma log(1 - P(e_{ij_k}) = 1))$$

The objective function is very similar to the one for the Skip-gram model using the negative sampling technique. After the gradient is obtained, the edge weight

w_{ij} still appears as a product of the product. The edge weight w_{ij} in the network varies widely. Therefore, affected by w_{ij}, the gradient will change greatly, which is still time consuming for training. This situation is also called gradient explosion and vanishing problem[19]. The edge sampling of Line algorithm is applied to solve this.



(a) negative sampling (b) edge sampling

Figure 3.5: negative sampling and edge sampling example in the network

3.2.3 Large-scale Information Network Embedding (LINE) and edge-sampling algorithm

There are mainly two outstanding contributions for LINE: one is that it can adapt to various types (undirected edges or directed edges, weighted and unweighted) large-scale (million-level nodes, billion-level edges) networks, and it can capture the first-order and second-order similarities in the network very well; the second is to propose a very powerful edge-sampling algorithm, which greatly reduces the time complexity of LINE. After using the edge-sampling algorithm, the time complexity and the number of edges in the network is linear. The efficiency of LargeVis also benefits from LINE and its edge sampling algorithm.

The first-order similarity refers to the point-to-point similarity between two nodes in the network, specifically the weight of the edge between the nodes (if the point does not have an edge, the first-order similarity is 0); the second-order similarity means that if nodes share similar neighbor nodes, then the two tend to be similar. For example, in the case shown in the figure 3.5(b), the weight of the edge is expressed in thickness.

The first-order similarity between node 8 and node 9 in the graph 3.5(b) is higher because the weight of the directly connected edges is higher. Node 1 and node 7

have most of the same neighbors, so the second-order similarity between the two is very high. The idea of edge sampling algorithm comes from the negative sampling optimization technique used by Mikolov in word2vec[4]. It does not only improves the efficiency of training, but also solves the problem of sharp increase in gradient caused by the weighted edge in the network representation during the training process.

As for the gradient explosion and vanishing problem that we face in the last section, it can be solved by the edge sampling technique in LINE.In fact, its principle is similar with negative sampling, but it is used in positive samples. If the weight of the edge between two points in the positive sample is w_{ij}, we can convert it into w_{ij} overlapping binary edges. For example, if the weight between node 8 and node 9 is 5, it will be converted into 5 binary edges with uniform weight.

If there are multiple large weighted edges (hundreds or even thousands), after converting to binary edges, the total number of edges is also very large, and all considerations will also affect efficiency. Therefore, after that all the weighted edges are converted into binary edges (equivalent to equidistant division), random sampling is performed from these binary edges (equivalent to weighted sampling). The advantages of the edge sampling algorithm are: on the one hand, because the sampled edges are binary edges, the weights are all the same, which solves the problem of a large range of gradient changes; on the other hand, the sampling process essentially follows a weighted sampling strategy. Because the edges with larger weights will convert more binary edges, the greater the probability of being sampled, which ensures correctness and rationality.

After using negative sampling and edge sampling optimization, LargeVis also uses asynchronous stochastic gradient descent for training. This technique is very effective on sparse graphs because the two nodes connected by edges sampled by different threads are rarely duplicated. There is almost no conflict between different threads. In terms of time complexity, the time complexity of each round of random gradient descent is $O(sM)$[16], where M is the number of negative samples, s is the dimension of the low-dimensional space (2 or 3), and the number of steps of the random gradient It is usually proportional to the number of nodes N, so the total time complexity is $O(sMN)$. From this, we can observe that the time complexity of LargeVis is linear with the number of nodes in the network.

Chapter 4

Uniform Manifold Approximation and Projection(Umap)

UMAP is another technique for dimensionality reduction using manifold learning after t-SNE and LargeVis. It comes from the theoretical framework of Riemannian geometry and algebraic topology. UMAP uses local manifold approximations and patches together their local fuzzy simplicial set representations to construct a topological representation of the high dimensional data[13]. and then a similar process can be used to construct an equivalent topological representation for the low-dimensional graph. So that it is optimized to be appropriately similar in structure. While ensuring scalability, it retains more of the global spatial structure and performs well during runtime[13]. At the same time, UMAP has no restrictions on the size of embeddings, and can be used as a general-purpose dimensionality reduction technology for machine learning. There are some differences from two other algorithms, leading to different performances:

4.1 Exponential probability distribution in high dimensions

UMAP uses a high-dimensional exponential probability distribution, but does not necessarily use Euclidean distance like t-SNE. Instead, it can insert any distance. In addition, it is different from the results obtained by t-SNE and LargeVis. Since the probabilities are not normalized.

$$p_{j|i} = \exp\left[-(r_{ij} - \rho_i)/\sigma_i\right]$$

Here ρ_i is the distance to the nearest neighbor (the zero distance repeated by

neighbors is not considered). An important parameter representing the distance from each $i-th$ data point to its first nearest neighbor. This ensures local connectivity of the manifold. In other words, this provides a locally adaptive exponential kernel for each data point, so the distance metric varies from point to point. At the same time, UMAP also uses a different symmetrization of high-dimensional probability, which is calculated by fuzzy set union using the probabilistic t-conorm. The probability can be expressed as:

$$p_{ij} = p_{i|j} + p_{j|i} - p_{i|j}p_{j|i}$$

Symmetrization is necessary because UMAP glues points with locally varying metrics together (through the parameter ρ), it may happen that the weights between nodes A and B are not equal to the weights between B and A nodes.

4.2 Probabilities in low dimensions

UMAP uses the curve family $1/(1 + a * y^{2b})$ to model the low-dimensional distance probability. It is not exactly the student t distribution, but it's very similar:

$$q_{ij} = (1 + a(y_i - y_j)^{2b})^{-1}$$

Where $a = 1.93$ and $b = 0.79$ represent the default value of the parameter(for $min\_dist = 0.01$). UMAP finds a and b from non-linear least-square fitting to the piecewise function with the min_dist hyperparameter.

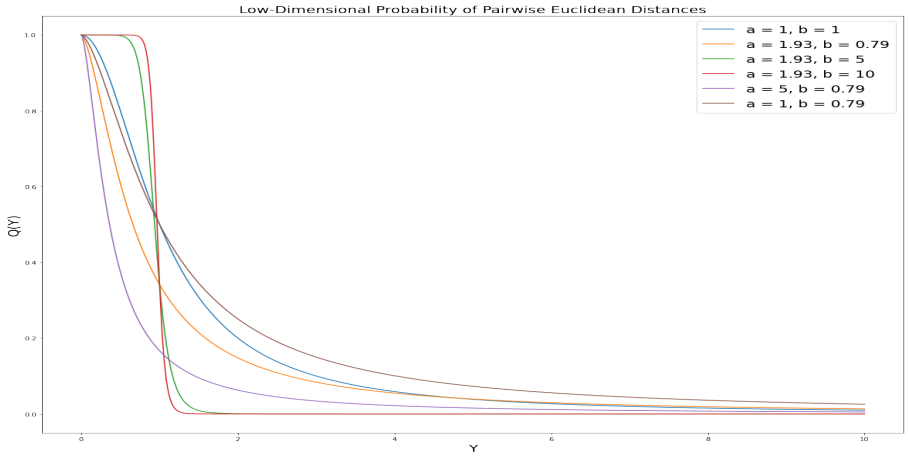


Figure 4.1: edge sampling example

The diagram 4.1 below shows how a and b change the curve. From the graph, we can easily find that a changes the smoothness of the slope, and b drastically

changes the shape of the function. After forming a smooth peak at the beginning of Y, it drops drastically, Which means when b is high, all data points are equally tightly connected in the beginning. That is, in the low-dimensional space, all the data points are similar in closeness and this may lead to the emergence of close clustering. UMAP uses the *min_dist* hyperparameter to find a and b from the nonlinear least squares fitting to the piecewise function:

$$q_{ij} = (1 + a(y_i - y_j)^{2b})^{-1} = \begin{cases} 1 & y_i - y_j \leq min\_dist \\ e^{-(y_i - y_j) - min\_dist}, & y_i - y_j > min\_dist \end{cases}$$

The algorithm does not apply normalization to high-dimensional and low-dimensional probabilities. As a result, the algorithm greatly reduces the time for calculating high-dimensional graphs, because summation or integration is a computationally expensive process.

4.3 Binary cross-entropy cost function

UMAP cost function is the cross entropy of two fuzzy sets, which can be expressed as a symmetric weight matrix:

$$C_{EUMAP}(X, Y) = \sum_i \sum_j [p_{ij}(X) \log(\frac{p_{ij}(X)}{q_{ij}(Y)}) + (1 - p_{ij}(X)) \log(\frac{1 - p_{ij}(X)}{1 - q_{ij}(Y)})]$$

Like the Kullback-Leibler divergence, this can be arranged into two constant contributions (those containing p_{ij} only) and two optimizable contributions (containing q_{ij}). Then, we can add the additional term in the C_{EUMAP} cost function which makes UMAP capable of capturing the global data structure:

$$C_{EUMAP}(X, d_{ij}) = \sum_j [-P(X) \log Q(d_{ij}) + (1 - P(X)) \log(1 - Q(d_{ij}))]$$

with

$$Q(d_{ij}) = \frac{1}{1 + a d_{ij}^{2b}} \qquad and \qquad \frac{\partial Q}{\partial d_{ij}} = -\frac{2abd_{ij}^{2b-1}}{(1 + a d_{ij}^{2b})^2}$$

Actually, the UMAP cost function has a very similar form with the one of LargeVis, but it does not have a γ term to weight the repulsive component of the cost function,

and does not require matrix-wise normalization in the high dimensional space. The cost function for UMAP can therefore be optimized with stochastic gradient descent in the same way as LargeVis.

$$\begin{aligned}\frac{\partial C_{EUMAP}}{\partial y_i} &= \sum_j \left[-\frac{P(X)}{Q(d_{ij})}\frac{\partial Q}{\partial d_{ij}} + \frac{1-P(x)}{1-Q(d_{ij})}\frac{\partial Q}{\partial d_{ij}} \right] \\ &= \sum_j \left[\left(-P(X)(1+ad_{ij}^{2b}) + \frac{(1-P(X))(1+ad_{ij}^{2b})}{ad_{ij}^{2b}} \right) \frac{\partial Q}{\partial d_{ij}} \right] \\ &= \sum_j \left[\frac{2abd_{ij}^{2(b-1)}P(X)}{1+ad_{ij}^{2b}} - \frac{2b(1-P(X))}{d_{ij}^{2}(1+ad_{i}^{2b}_{j})} \right] (y_i - y_j)\end{aligned}$$

In contrast to the random normal initialization used by t-SNE, UMAP uses the graph Laplacian algorithm to assign initial low-dimensional coordinates. However, this should not have much impact on the final low-dimensional representation, at least for t-SNE[13]. In fact, this will reduce UMAP changes between each run because it is no longer randomly initialized. The choice of initialization by graph Laplacian is to minimize KL divergence with early exaggeration in the initial stage of t-SNE is equivalent to constructing Graph Laplacian. The graph Laplacian actually combines matrix factorization and neighbor graph methods to solve the DR problem. In this method, we first construct a knn-graph, then form the Laplacian matrix with matrix algebra, and finally decompose the Laplacian matrix to solve the eigenvalue decomposition problem[24].

Chapter 5

Assessment Algorithm

Similarity-based methods using Stochastic Neighbor Embedding such as t-SNE and its variants Umap and Largevis emerged with remarkable DR performances and also the time complexity. As the method to compare the performance for the result of algorithms. This chapter focuses on exploring DR quality measurement criteria.

There are some studies that developed DR quality criteria measuring the HD neighborhoods preservation in the LDS [7], which becomes generally adopted. The quality criterion used to assess the various embedding evaluates the preservation of K nearest neighborhoods[1]. This work examines the DR quality of these algorithm on numerous real-world and artificial data sets[5], using various potential parameter values for each one, given by neighborhood-based DR performance criteria [6]. With v_i^K and n_i^K index the K nearest neighbors of high dimension data point x_i and y_i in the HD and LDS, we can define:

$$Q_{NX}(K) = \frac{\sum_{i \in I} \mid v_i^K \cap n_i^K \mid}{KN} \in [0, 1]$$

This criterion measures the average normalized agreement between corresponding K nearest neighborhoods in the HD and LD spaces[14]. Its expectation for random LD points becomes $K/(N-1)$. With the formula above, we can define:

$$R_{NX}(K) = ((N-1)Q_{NX}(K) - K)/(N-1-K)$$

R_{NX} is to compare different neighbour sizes [8]. For K $\in [1, N-2]$. R_{NX} equal to -1 means that instead of collecting nearest neighbours, the result regards furthest neighbors on opposite. The value 0 means that this is a random embedding($R_{NX} \approx K/(N-1)$)[1]. When R_{NX} equal to 1, all the K nearest neighbors fit are exactly the closest ones. The curve of the DR quality could be displayed with a log-scale for K as closer neighbors. The area under the DR quality measurement curve could be expressed as:

$$AUC = \frac{\sum_{K=1}^{N-2} R_{NX}(K)/K}{\sum_{K=1}^{N-2} K^{-1}} \in [-1, 1]$$

This work is going to use AUC value as the measurement for performance of the DR results. The higher AUC implies the higher R_{NX} in all K field, which represents a better performance on the dataset.

Part III

Comparison

Chapter 6

Comparisons and implementations

This work will analyze the characteristics of the three algorithms on different real-world data sets. Taking into account the differences between instances (N) and dimension (M), 6 data sets with different N and M will be tested. The code will test the AUC values of the algorithms, and first analyze the individual parameters. Then, analyze the parameters that affect each other according to the generated heatmap. Finally, this work will take 5 representative values from each variable of each algorithm, use grid search algorithm to find the parameters that each algorithm performs best on the same data set. Analyze the performance difference of R_{NX} curve.

As discussed above, in order to draw the conclusions in a convincing way, This work will examine the three algorithms' quality on several real-world data sets with different N and M, based on the neighborhood-based DR performance criteria discussed above. There are six public databases: Iris, Wine, Breast Cancer Wisconsin Diagnostic (BCW)[25], the Olivetti faces (Oliv), Optical Recognition of Handwritten Digits test set (Digits) and Labeled Faces in the Wild face recognition(LFW)[26]. The instance and dimension information of datasets are shown on table 6.1.

6.1 Grid search algorithm

Grid search is a parameter adjustment method, with exhaustive strategy: among all the predetermined parameter selections, through looping and trying every possibility, the best performing parameter is the final result. Take this thesis's work as an example. Each algorithm selects the four most important parameters,

Datasets	Number of instances	Number of dimensions
Iris	150	4
Wine	178	13
Oliv	400	4096
BCW	569	30
Digits	1797	64
LFW	13233	5749

Table 6.1: instance and dimension information of datasets

and each parameter takes 5 typical values. All possibilities are listed, which can be expressed as a 5*5*5*5 four-dimensional table. Each numerical combination is a grid, and the cycle process is like traversing and searching in each grid, so it is called grid search. As for this work, the code will go though all combination of parameters and return the best parameters considering neighborhood-based DR performance criteria.

6.2 BH t-SNE

As discussed above in chapter 2, since the result from t-SNE and BH t-SNE have similar result while BH t-SNE is much faster than the other[6]. This thesis will focus the performance and characteristics of it in the experiment part. There are 4 most important parameters for BH t-SNE, the ideas of each parameters are:

1. *perplexity*: related with the number of nearest neighbors for each point
2. *n_iter*: maximum number of iterations for optimization
3. *early_exaggeration*: the tightness inside clusters and the distance inbetween
4. *min_grad_norm*: the threshold for stopping the optimization

6.2.1 *perplexity*

perplexity is related to the number of nearest neighbors. It changes the balance between the local and global aspects of the data. The *perplexity* value has a complex effect on the generated pictures, but the performance of SNE is quite reliable for changes in complexity, and the typical value of perplexity is between 5 and 50[6].

We can try different potential perplexity values on datasets with different instances. At the same time, since the algorithm using stochastic method, the parameter *random_state* is set to 1. This parameter determines the random number generator so that it pass an int mark for reproducible results across multiple function calls. After examining it on the first dataset Iris, we get the figures below. For comparing visualization quality on the dataset, from this result, the experiment colored each node with the information of its label in the visualized low-dimensional graph:

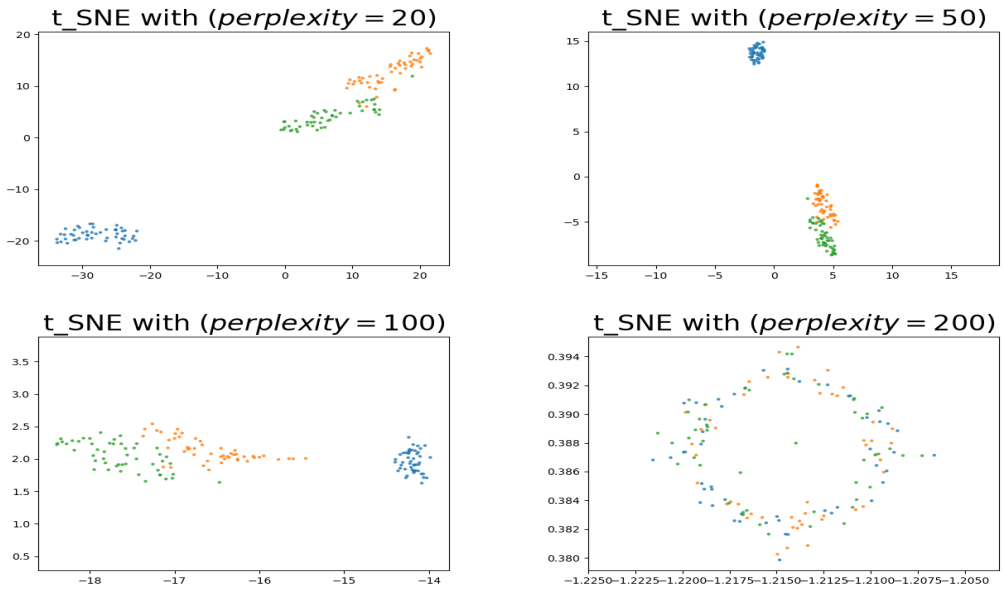


Figure 6.1: LD result for BH t-SNE with different perplexity on Iris

In the figure, we can see that when the value is in the range of 5-50, clusters are well distinguished. Considering that the number of instances of the dataset is only 150, when the value is 100, the clusters begin to merge. When the perplexity equals to 200, it is difficult to distinguish different clusters. Considering that the *perplexity* varies from dataset to dataset, it is better to modify the parameter depending on each specific dataset. Loosely speaking, one could say that a larger/denser data set requires a larger perplexity[20].

At the same time, the AUC value for the result with these parameters are 0.731, 0.658, 0.654, -0.004 separately. These AUC values verify the previous observations. As for the last one, it has a big difference between others as a minus result, which means that the result is bit worse than random embedding. With the definition of the R_{NX} curve $R_{NX}(K) = ((N-1)Q_{NX}(K) - K)/(N-1-K)$. This is because perplexity is set to a high value, the Q_{NX} on the numerator is relative small since

the normalization term is largest and denominator is a negative number, we then have a minus result.

6.2.2 *n_iter*

n_iter is the maximum number of iterations for the algorithm. The graph observed above are all generated with 1000 iteration which is the default value of *n_iter*. In principle, the higher *n_iter* gives the better result. However, it may be very time consuming when it comes to big dataset. In contrast, if *n_iter* is too small, the algorithm does not have enough iteration to generate the convincing clusters in LD. The most important thing is to reach a stable configuration. The figure shows how the result changes with *n_iter* grows in Digits dataset as above:

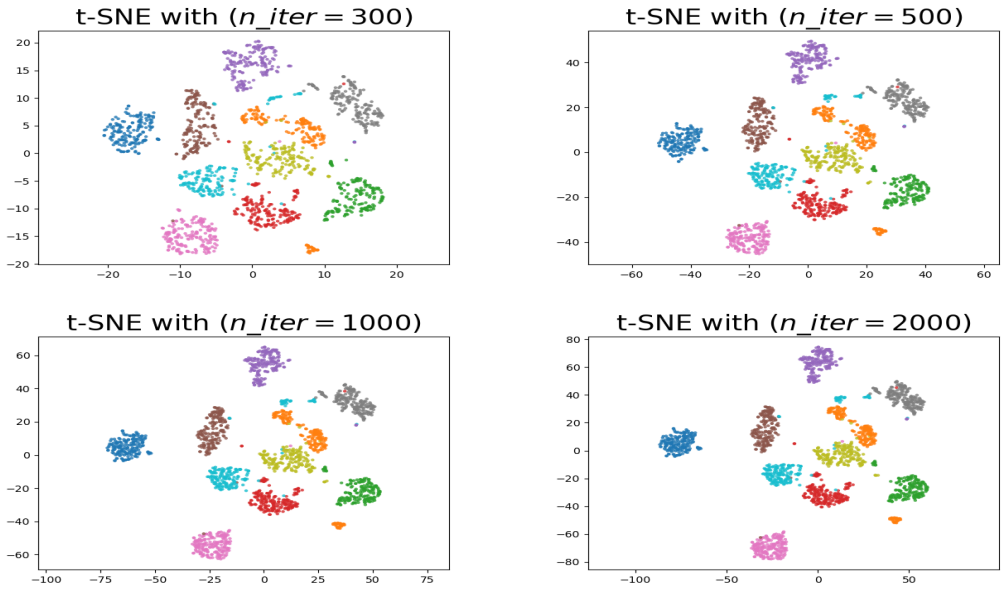


Figure 6.2: LD result for BH t-SNE with different n_iter on Digits

From the figure, we can easily observe that the more iterations, the better identified for clusters. These four *n_iter* generate *AUC* values are 0.489, 0.526, 0.533, 0.536 separately, which verified the previous conclusion.

6.2.3 *min_grad_norm*

This parameter is the threshold of gradient norm to determine when the optimization will be stopped. If the norm is less than the setting value, the gradient descent of Kullback-Leiber divergence will stop. Considering that the default number of this

parameter is small enough, after examining its potential values, the accuracy is promised if it is set to a relative small value. When it becomes too big, gradient descent may not even find out the local minimum.

6.2.4 *early_exaggeration*

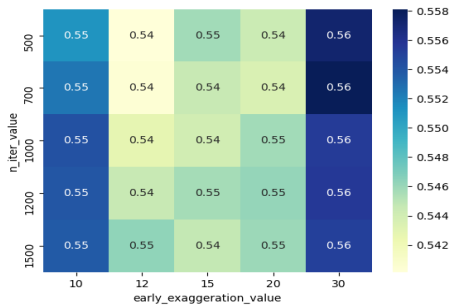
early_exaggeration is equivalent to increasing perplexity for a short while. It controls how tight natural clusters in the original space are in the embedded space and how much space will be between them. When the value becomes higher, the space between clusters will be larger in the embedded space. It also becomes easier for the clusters to move around relative to one another in order to find a good global organization[9].

6.2.5 Relations between parameters

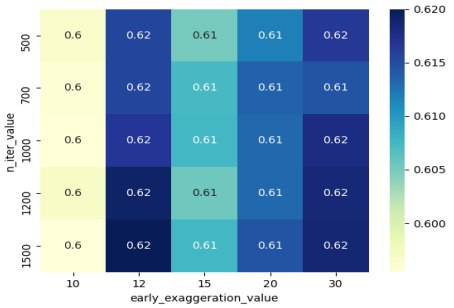
The code of examination also generates a series of heatmaps to show the relationship of each parameter pair. Heatmap represents the effect of one parameter change on another parameter when other parameters take default values. According to the operation of BH t-SNE on 6 data sets, :

early_exaggeration **and** *n_iter*:

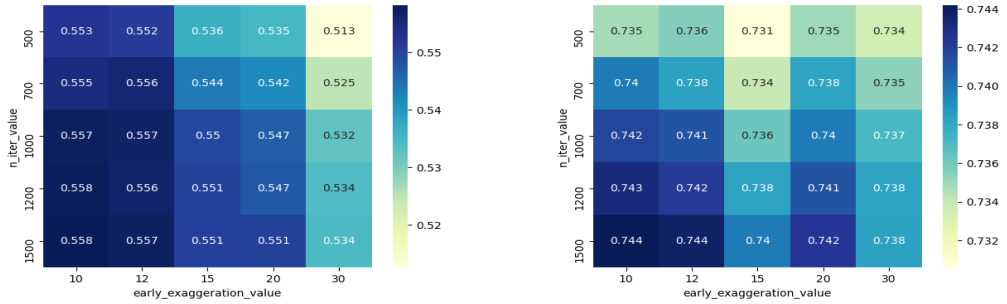
The figures below shows the heatmap between *early_exaggeration* and *n_iter* for each dataset:



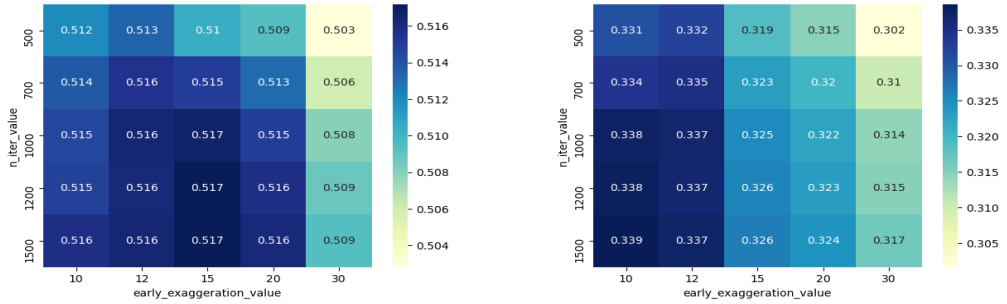
(a) Iris



(b) Wine



(c) Oliv (d) BCW



(e) Digits (f) LFW

Figure 6.3: *early_exaggeration* and *n_iter* for BH t-SNE on different dataset

It is easy to find that when *early_exaggeration* goes smaller and *n_iter* goes larger, the performance of BH t-SNE is the better. When these two parameters come to the smallest and the largest of the range separately, the AUC value is the best. Except on two small dataset (Iris and Wine), where the best result is almost as good as other values with the difference of AUC to other values less than 0.01. This makes sense since larger *early_exaggeration* force clusters to move around relative to one another in order to find a good global structure. With the number of iteration increasing, there are higher probability for algorithm to reach a stable configuration.

Running time

As in the previous theoretical analysis, It is easy to find that running time increase with *perplexity* and *n_iter* affect the running time the most. The time increases significantly with these two parameters independently with the same trend on 6 data sets. Take digits dataset as an example, there are heatmaps for these two parameters with others:

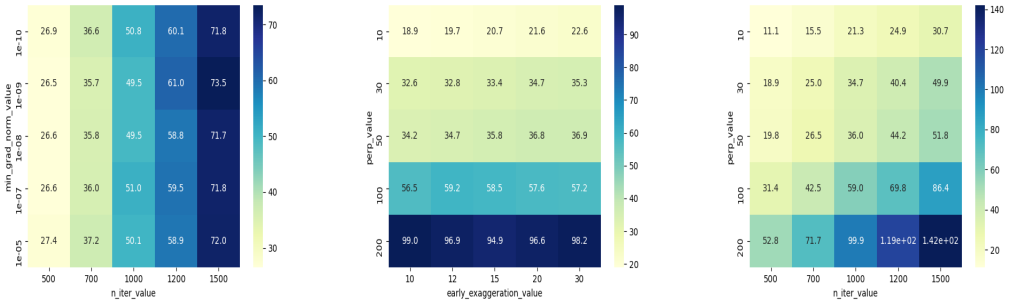


Figure 6.4: Running time for perplexity and n_iter on Digits dataset with other parameters set to default

6.3 UMAP

There are the four most important parameters for UMAP, the ideas of each parameters are:

1. *n_neighbors*: The size of local neighborhood like perplexity for BH t-SNE
2. *min_dist*: The effective minimum distance between embedded points
3. *spread*: The effective scale of embedded points
4. *negative_sample_rate*: Number of negative samples for per positive sample

6.3.1 *n_neighbors*

n_neighbors determine the size of the local neighborhood used for manifold approximation. Its default value is 15 instead of 30 for *perplexity* in BH t-SNE. This makes sense since the definition of K is $k = 2^{\sum_i p_{ij}}$ where $p_{ij} = p_{i|j} + p_{j|i} - p_{i|j}p_{j|i}$ without the log 2 function in perplexity as $perplexity = 2^{-\sum_i p_{i|j} \log_2 p_{i|j}}$.

The changes allow this parameter controls how UMAP balances local versus global structure in the data. It does this by constraining the size of the local neighborhood. UMAP will look at when attempting to learn the manifold structure of the data. This means that low values of *n_neighbors* will force UMAP to concentrate on very local structure, while large values will push UMAP to look at larger neighborhoods of each point when estimating the manifold structure of the data, losing fine detail structure for the sake of getting the broader of the data. Also, since the parameter *n_neighbors* means the number of the neighbors, it should not be higher than the

number of data points.

We can try different potential perplexity values in datasets with different instances. Same as BH t-SNE, in order to avoid the effect of stochastic for each round, the parameter is $random_state is$ set to 1. After examining it on the dataset Digits, we get the figures below:

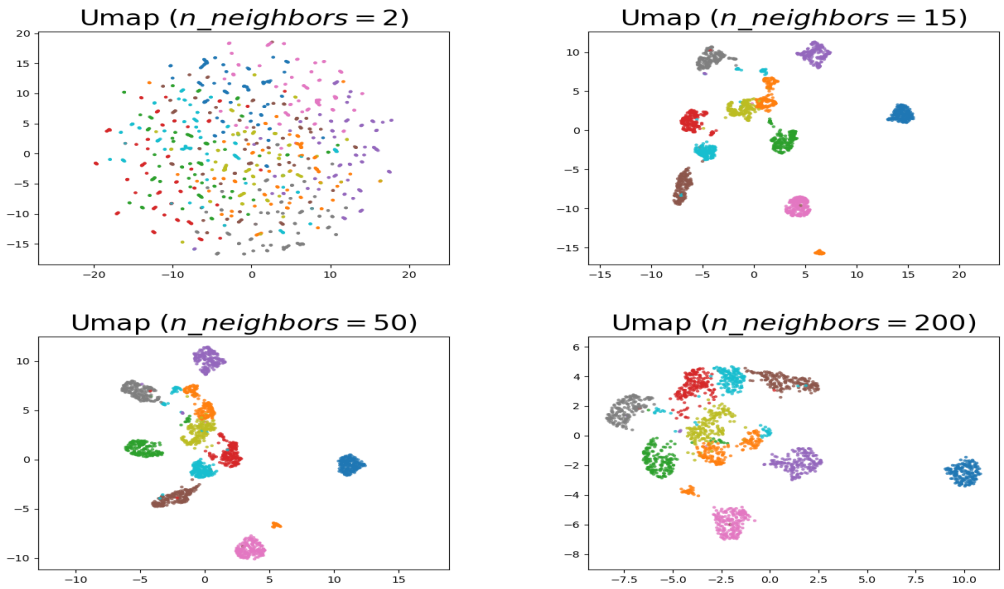


Figure 6.5: LD result for UMAP with different n_iter

These values of *n_neighbors* lead to AUC values for 0.391, 0.449, 0.437, 0.431 seperately. As we can observe from the figure, with the number of *n_neighbors* increases, when constructing graphical representations of high-dimensional data, UMAP connects more and more adjacent points, which leads to a projection that more accurately reflects the global structure of the data. When it is a very low value at 2, it focus on the local structure completely.

6.3.2 *min_dist*

The min_dist parameter controls how tightly UMAP is allowed to pack points together. It, quite literally, provides the minimum distance apart that points are allowed to be in the low dimensional representation. This means that low values of *min_dist* will result in clumpier embeddings. Larger values of *min_dist* will prevent UMAP from packing points together and will focus instead on the

preservation of the broad topological structure.

UMAP uses the family of curves $1/(1+a*y^{(2b)})$ for modelling distance probabilities in low dimensions, not exactly Student t-distribution but it is very similar, without normalization. As the formula mentioned in the theoretical part, the *min_dist* determine the a and b for the curve directly. Here below are figures with different *min_dist* with the Digit dataset as above:

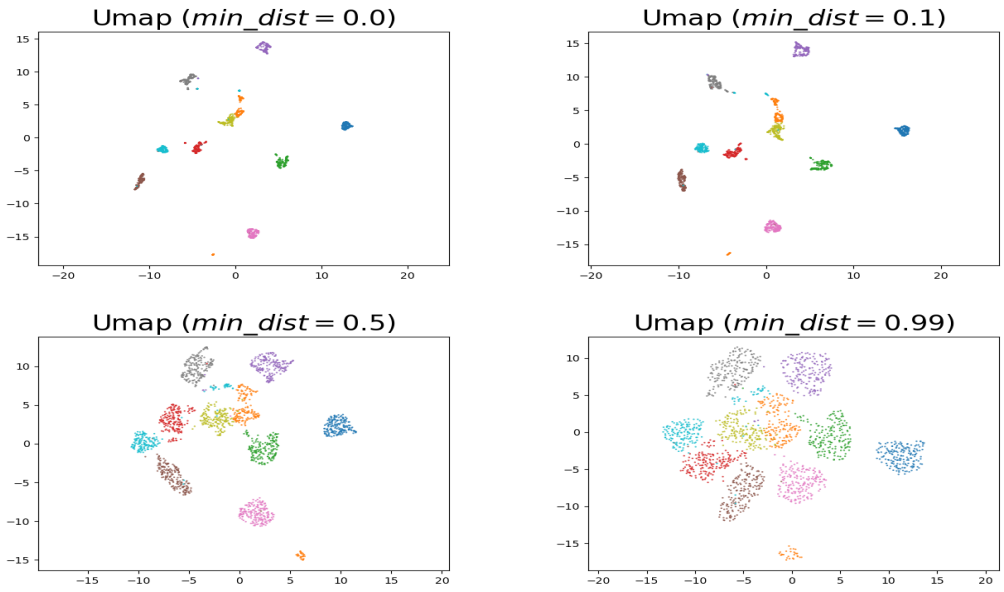


Figure 6.6: LD result for UMAP with different min_dist

From the figures above, we see that using *min_dist*=0.0, UMAP can find smaller connected components and clusters in the data, and emphasize these features in the final embedding. As these structures in *min_dist* increase, the size of the clusters also increase. These structures are pushed into softer and more general functions, which provide a better overall view of the data without the loss of more detailed topology.

6.3.3 *spread*

spread control the effective scale of embedded points, which means that it determines the scale at which embedded points will be spread out. Whereas increasing spread keeps the shape and boundary of the clusters a bit better. Spread can therefore be used to control the inter-cluster distances to some extent, where as *min_dist* controls the size of the clusters.

6.3.4 *negative_sample_rate*

This parameter determine the number of negative samples to select per positive sample in the optimization process. Increasing this value will result in greater repulsive force being applied, greater optimization cost, but slightly more accuracy. Here below are figures for different *negative_sample_rate* values:

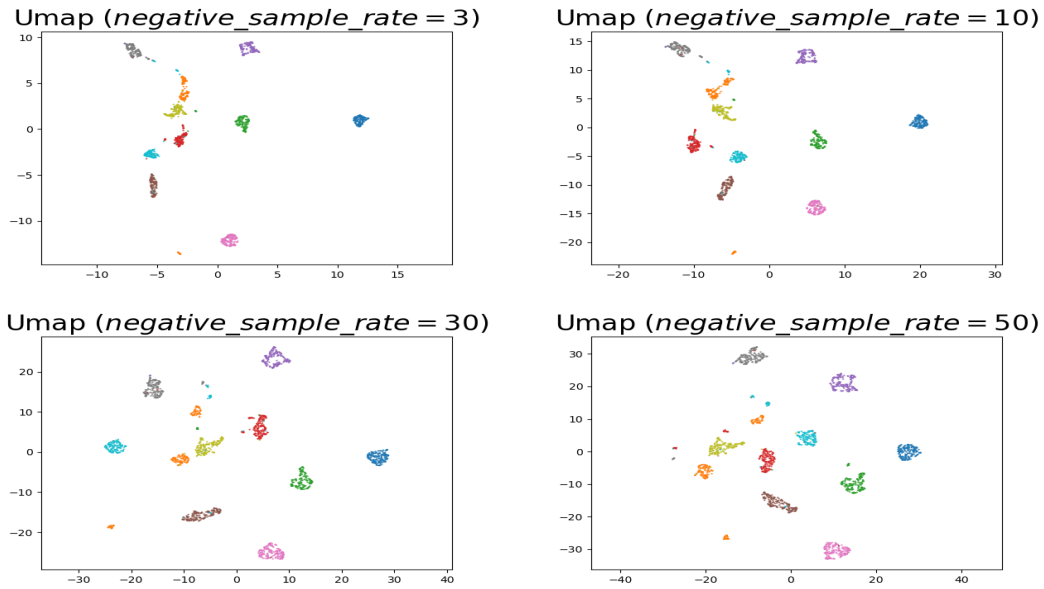


Figure 6.7: LD result for UMAP with negative_sample_rate

As we can observe, the clusters are more well identified with the increase of *negative_sample_rate*. As for the result, these four settings of the parameter has *AUC* values 0.402, 0.425, 0.430, 0.449. At the same time, the running time for each are 4.385, 7.749, 9.552, 25.615, 45.477 separately, which confirmed the previous argument.

6.3.5 Relationships between algorithms and parameters

Running time

The most obvious finding is that UMAP uses much less time than BH t-SNE to run the code in each dataset. Here below is the comparison between UMAP and BH t-SNE:

There are several reasons for this:

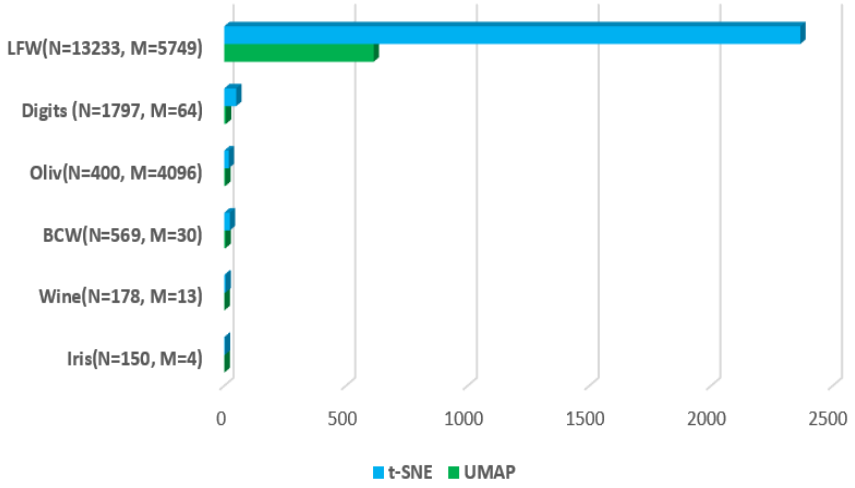


Figure 6.8: Time consumption on different dataset for 2 algorithms

1. As defined above, the High-dimensional and low-dimensional probability is expressed as $p_{i|j} = e^{-\frac{d(x_i,x_j)-\rho}{\sigma_i}}$ with $p_{ij} = p_{i|j} + p_{j|i} - p_{i|j}p_{j|i}$ and $q_{ij} = (1 + a(y_i - y_j)^{2b})^{-1}$, which do not apply normalization. Although they have been scaled for the segment [0,1], it turns out that there is no normalization, such as the denominator in the equation of $p_{i|j}$. The time for calculating high-dimensional graphs is greatly reduced, because summation or integration is a computationally expensive process.

2. Different with the random normal initialization used by tSNE, UMAP uses the graph laplacian method to assign initial low-dimensional coordinates. This will reduce the UMAP change from iteration to iteration because it is no longer random initialization.

The global structure preserved better using UMAP

With a better global structure visualization result and better AUC acore, UMAP has a better balance between local and global structure.[21, 22]

This firstly because of the different cost function using Cross-Entropy instead

of the KL-divergence. As discussed above, we have:

$$C_{EUMAP}(X, d_{ij}) = \sum_j [-P(X) \log Q(d_{ij}) + (1 - P(X)) \log(1 - Q(d_{ij}))]$$

It can turn out to be:

$$\begin{aligned} C_{EUMAP}(X, d_{ij}) &= \sum_j [-P(X) \log Q(d_{ij}) + (1 - P(X)) \log(1 - Q(d_{ij}))] \\ &= e^{-X^2} \log \left[e^{-X^2} (1 + Y^2) \right] + (1 - e^{-X^2}) \log \left[\frac{(1 - e^{-X^2})(1 + Y^2)}{Y^2} \right] \\ &\approx e^{-X^2} \log(1 + Y^2) + (1 - e^{-X^2}) \log(\frac{1 + Y^2}{Y^2}) \end{aligned}$$

This leads to the change in the preservation balance of the local-global structure. At a smaller value of X, we obtain the same limit as t-SNE, because the second term disappears due to the previous factor, and the logarithmic function is slower than the polynomial function:

$$X \to 0 : CE(X, Y) \approx \log(1 + Y^2)$$

Therefore, in order to minimize the loss, the Y coordinate is forced to be small, that is, $Y \to 0$. This is exactly the same behavior as t-SNE. However, in the opposite limit of large X, the first term disappears and the former factor of the second term becomes 1, we get:

$$X \to \infty : CE(X, Y) \approx \log(\frac{1 + Y^2}{Y^2})$$

If Y is small, we will get a high penalty due to Y in the logarithmic denominator. The larger Y is encouraged so that the ratio under the logarithm becomes 1, and we get zero penalty. Therefore, we get $Y \to \infty$ at $X \to \infty$, so when moving from high-dimensional space to low-dimensional space, the global distance is preserved.

Relationship between *perplexity* and *n_neighbor*

BH t-SNE and UMAP both define the high-dimensional probability at a certain distance for observing points as:

$$p_{ij} \approx e^{-\frac{(x_i - x_j)^2}{2\sigma_i^2}}$$

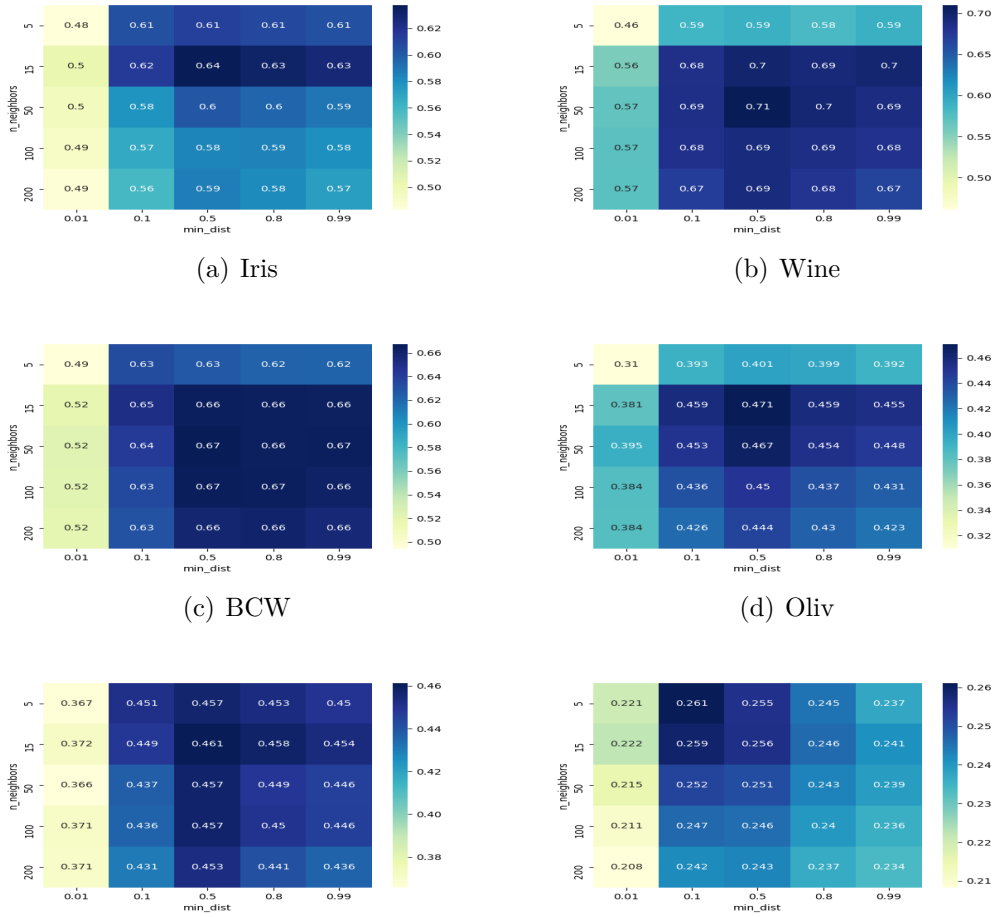
Here σ is the parameter for how many samples can be detected by each other, which is a finite value. This means that the data points over the threshold of σ will not be considered as a candidate to analyze. Because both tSNE and UMAP are neighbor graph algorithms, the local structure of the graph is retained. However, when $\sigma \to \infty$, every point has a chance to detect every other point, which means, in principle, both BH t-SNE and UMAP can retain the global structure. Even thoughr σ is not the hyperparameter of tSNE and UMAP, but as a variable of *perplexity* and *n_neigbor* respectively.

We can easily observe that the *n_neighbor* hyperparameter is increased, the average sigma of UMAP will soon reach a plateau, while BH t-SNE is much more sensitive to *perplexity*. When it comes to big *perplexity*, the almost hyperbolic difference of the average σ of BH t-SNE has a huge impact on the gradient of the tSNE cost function (KL difference). In the limit $\to\infty$, the high-dimensional probability in the above formula becomes 1, which leads to a decrease in the KL divergence gradient.

Relationship between *n_neighbors* **and** *min_dist*

From the previous derivation, we can draw the conclusion that *n_neighbors* change the balance between local and global structure and *min_dist* determine the size of the clusters. These two parameters affect the performance of the algorithm the most.

The figures below show the heatmap between *n_neighbors* and *min_dist* for each dataset, it is easy to find that when *min_dist* goes to middle of the range around 0.5 and *n_neighbors* also goes to middle of the range around 50, the performance of UMAP is the best. Except for LFW dataset, where the best result comes from two parameters are set to almost the minimum of the range. This situation happens because the size of LFW is 10 to 100 times higher in both instances and dimensions comparing with the rest of the dataset. So the algorithm need to focus more on smaller scale of the structure and keep the size of the clusters relatively small in order to have a better dimension reduction result.



(a) Iris (b) Wine

(c) BCW (d) Oliv

(e) Digit (f) LFW

Figure 6.9: Heatmap result with different *n_neighbors* and *min_dist* in different dataset

Influence of parameters on running time

Although UMAP runs faster than BH t-SNE, there are still differences on how each parameter infect the time consumption. *n_neighbors* and *negative_sample_rate* are two most decisive parameters with the increasement of them. Due to space limitations, this scheme will show the time consumption tendency in the heatmap form Digits, which is the representative dataset:

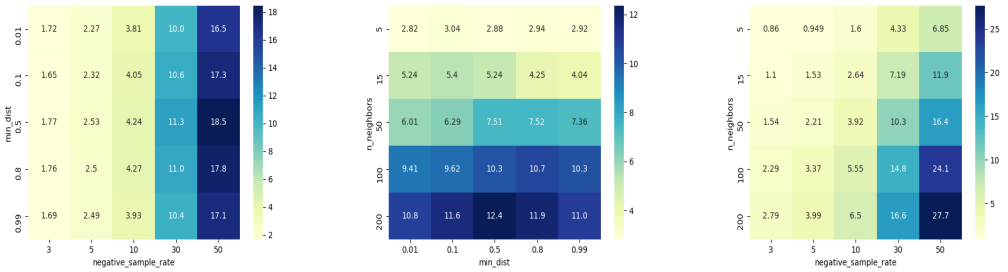


Figure 6.10: Running time for negative_sample_rate and n_neighbors on Digits dataset with other parameters set to default

It is clear that *n_neighbors* and *negative_sample_rate* play a major role in the increase of running time. When it comes to heatmap for *n_neighbors* and *negative_sample_rate*, the highest running time comes when these two parameters set to highest.

6.4 Largevis

This experiment focuses on the four most important parameters of the Largevis algorithm. Their basic ideas are:

1. *perp*: The perplexity used for deciding edge weights in KNN graph
2. *neg*: Number of negative samples used for negative sampling
3. *gamma*: The weights assigned to negative edges
4. *neigh*: Number of neighbors (K) in KNN graph, which is usually set as three times of perplexity

6.4.1 Installation and modification

Because the Largevis algorithm is not integrated in any python software package, but a series of C++ and python source codes are provided on github, it needs to be deployed before using the algorithm.

The C++ code needs to be compiled in the VS environment, so the first step is to download and configure Visual studio. Second, Since a large number of related library functions are used in the project, Boost must be configured in advance. The Boost library is a portable C++ library that provides source code. As a

backup to the standard library, it is one of the development engines of the C++ standardization process.

In principle, the Largevis should be installed on the computer after two procedures. However, since the python code of the source code is based on python 2.7 and version on the computer is 3.7. Also the version of C++ code is different, there are some API that do not use the previous names and the usage of some functions changed also. It is necessary to modify the code base on the differences as well.

6.4.2 *perp*

Since the approach to calculate conditional probability in HD and the weights of KNN graph are the same as BH t-SNE, the parameter *perplexity* play the similar role. This is basically to maintain a balance between the local and global aspects of the data by deciding edge weights in building HD KNN graph. Here below are visualization results with different potential perplexity values on 6 datasets:

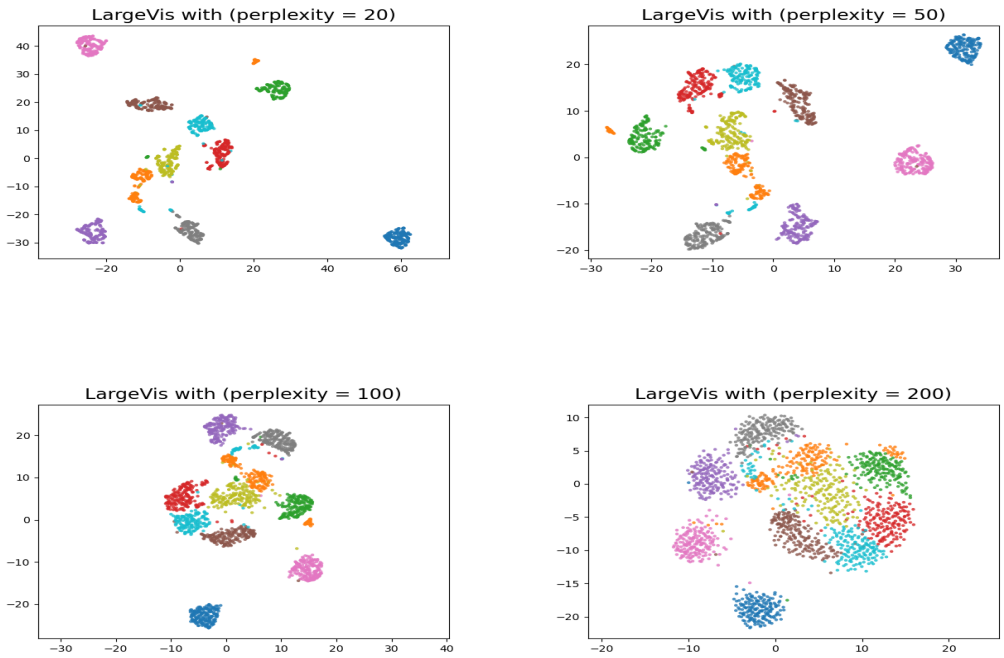


Figure 6.11: LargeVis results with different Perplexity

From the figure we can observe that when the *perplexity* is 20 and 50, the clusters are well separated with a better global and local balance. While after 100, clusters start to merge. When *perplexity* reach 200, the gap in between are blurry. Because

there are not the parameter $random_stateis$ in its source code like t-SNE and UMAP, the AUC values are average score with 5 iterations for each in order to avoid randomness. The AUC values of them are 0.48, 0.487, 0.375, 0.383 separately.

6.4.3 *neigh*

neigh represents the number of neighbors in K-NN graph. With higher *neigh* values, the more neighbors will be detected when building KNN graph. As with the official BH t-SNE implementation, the LargeVis reference implementation uses a default perplexity of 50, and the default number of nearest neighbors is 3 times the perplexity.

6.4.4 *neg*

As the key parameter for negative sampling in LD visualization part. *neg* determines the number of negative samples used. Same as for the theoretical part, if we select an edge in the set of edges with the non-zero weight(E) from the KNN graph, so that $p_{ij} \neq 0$. This is called a "positive edge". i and j are used to calculate the attractive part of the gradient. If we sample one of the N vertices(*neg*). As datasets grow larger, the probability that *neg* is in E grows smaller. Here below are visualization results with different potential perplexity values on Digits dataset:

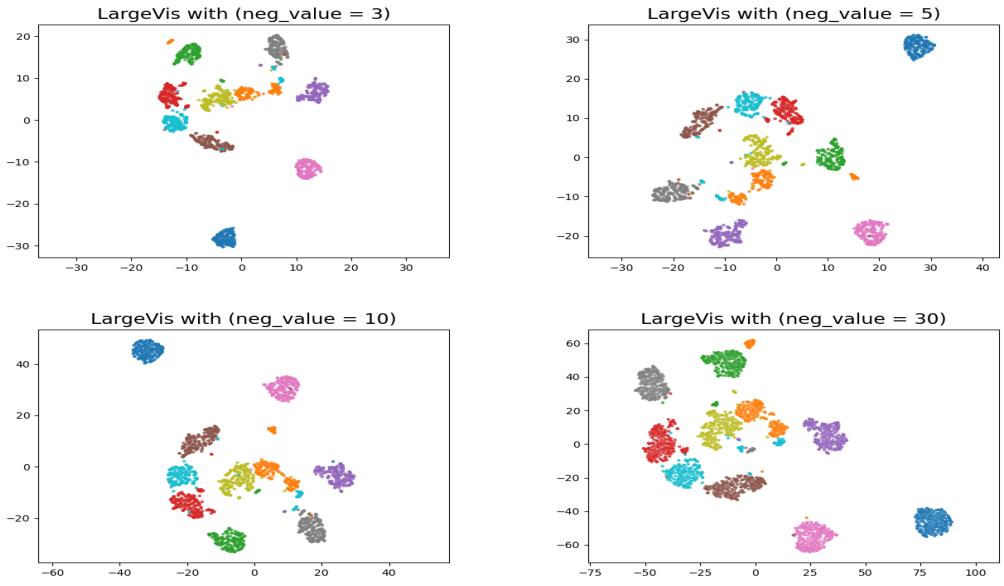


Figure 6.12: LargeVis results with different neg

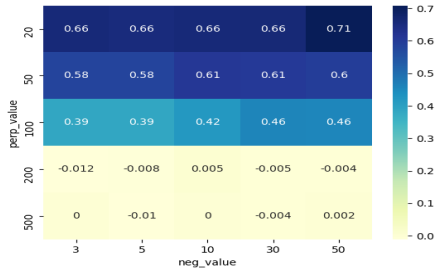
As we can see from 6.13, the average distance between clusters grows as *neg* variable increases. Their AUC values are 0.45, 0.46, 0.50, 0.51 separately. This is because the "repulsive force" of the cost function strengthened. At the same time, there are more time consumption since the vertices from KNN graph need to consider increased multiple times.

6.4.5 *gamma*

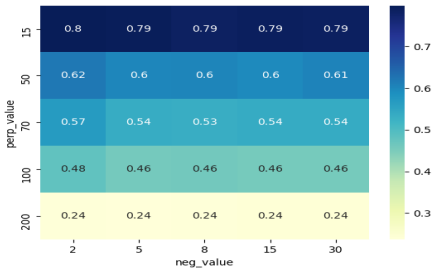
gamma stands for the weights assigned to negative edges. In the optimization process of Largevis, the goal is to maximize the probability that a positive sample node pair has a connected edge in the KNN graph, and to minimize the probability that a negative sample node pair has a connected edge in the KNN graph, where γ is the unified the weight set by the negative sample edge. Similar to the relationship between perp and neigh, *nag* and *gamma* set the number and weight of negative sample edges.

6.4.6 relationships between parameters

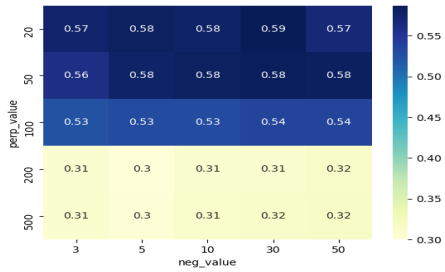
After observing the heatmap results, the two most important parameters *perp* and *neg* for building KNN graph in HD and visualization algorithm in LD, have the greatest impact on AUC values. Here below are the relationships between them:



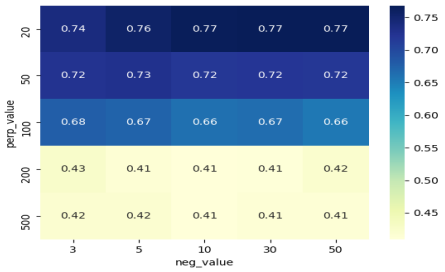
(a) Iris



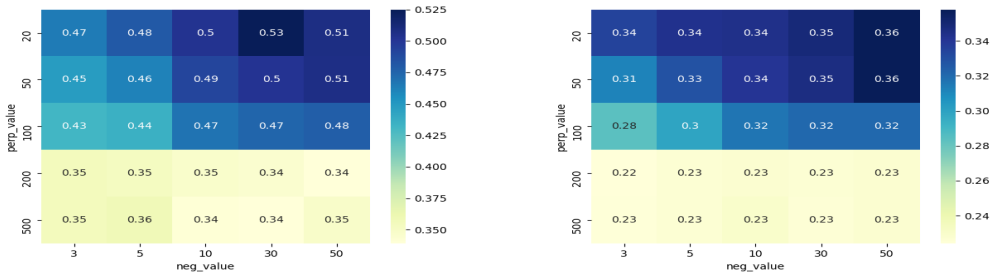
(b) Wine



(c) Oliv



(d) BCW



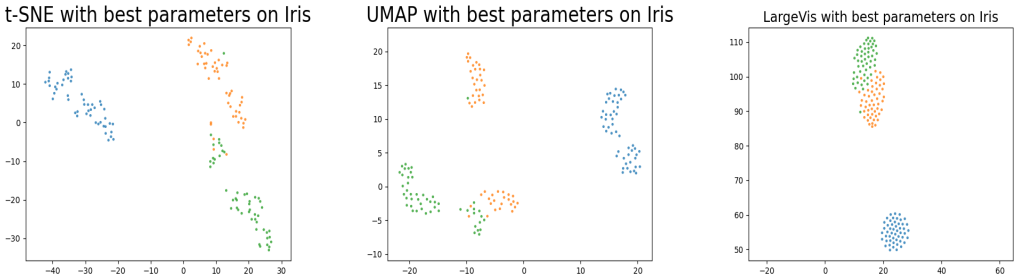
(e) Digits (f) LFW

Figure 6.13: heatmap between *perp* and *neg* on each datasets

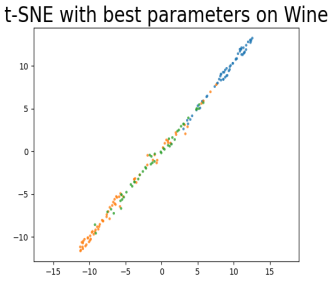
With *perplexity* increases, the performance becomes worse on average. when the range of neighbor becomes too much, the redundant information from less relative neighbor may damage accuracy. While higher *neg* leads to better result. Because the bigger the number of negative samples, the greater the probability that the node pair of the positive sample has a connected edge in the KNN graph, the smaller the probability that the node pair of the negative sample has a connected edge in the KNN graph. This lead to better performance with more accurate sampling.

6.5 Visualization result with optimal parameters

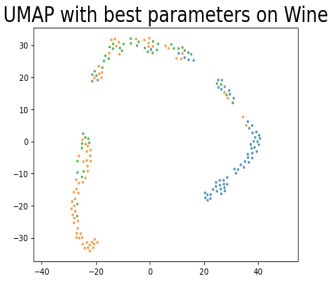
After examining all representative representative parameter values using grid search, the optimal parameter settings of each algorithm for each dataset are generated. The figure below shows the visualization results of each algorithm on the data set:



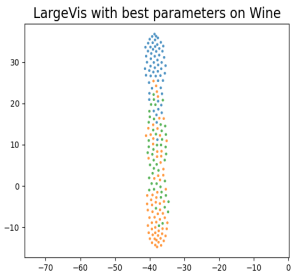
(a) BH t-SNE on Iris (b) UMAP on Iris (c) LargeVis on Iris



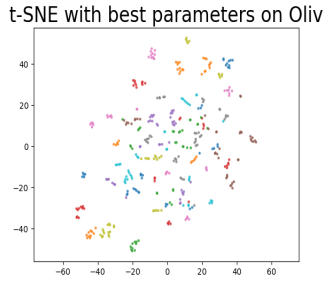
(d) BH t-SNE on Wine



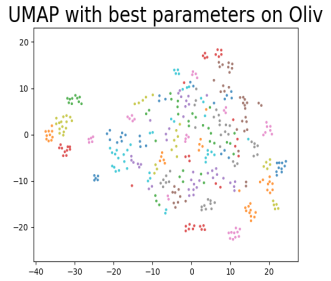
(e) UMAP on Wine



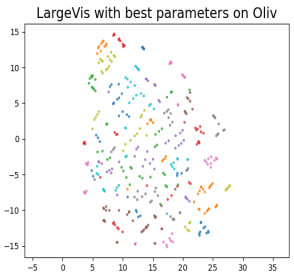
(f) LargeVis on Wine



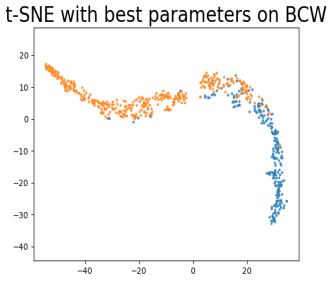
(g) BH t-SNE on Oliv



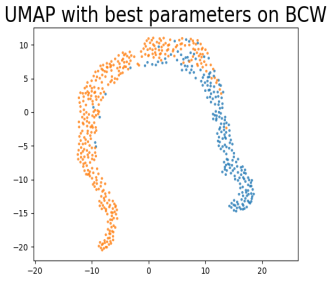
(h) UMAP on Oliv



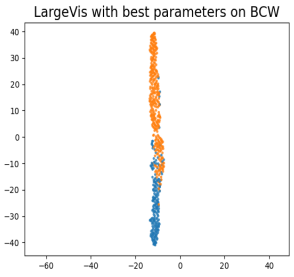
(i) LargeVis on Oliv



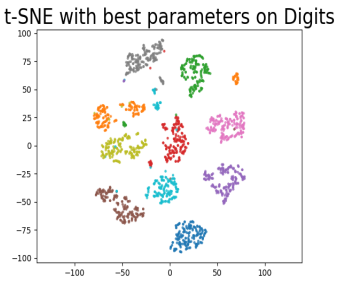
(j) BH t-SNE on BCW



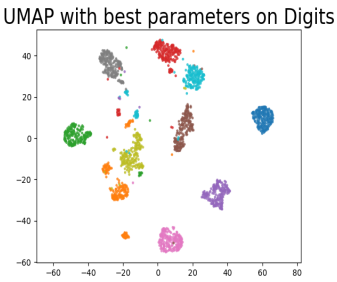
(k) UMAP on BCW



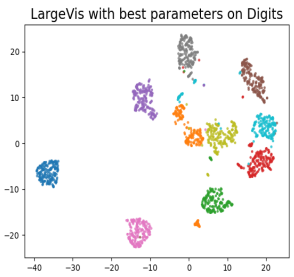
(l) LargeVis on BCW



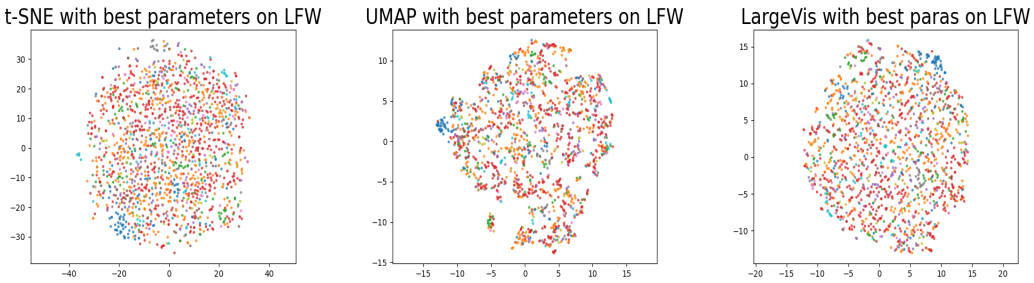
(m) BH t-SNE on Digits



(n) UMAP on Digits



(o) LargeVis on Digits



(p) BH t-SNE on LFW (q) UMAP on LFW (r) LargeVis on LFW

Figure 6.14: Visualization results on 6 datasets

We can see that on the smallest data set with hundreds of instances and dimensions, like Iris, Wine and BCW, the visualization images generated by BH t-SNE, UMAP and LargeVis are both meaningful and can be compared with each other.

However, when it comes to the image dataset, like Oliv(with N = 400, M = 1096) and LFW(with N = 13233 , M = 5749), all three algorithms do not have a good performance. At the same time, the outcome of Digits dataset(with N = 1797, M = 64), which is also a image dataset, identify the clusters clearly. This may because the image of the first two datasets are consist of hundreds of objects at different angles, after transforming them into pixels, the image of same object at different angle could be regarded as different clusters. The following chart and table show the AUC values and time consumption on each dataset of three algorithms with optimal parameters:

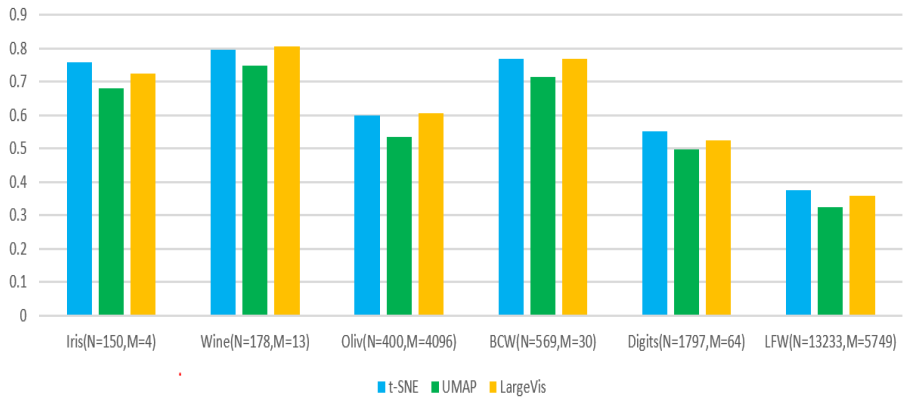


Figure 6.15: AUC values for each dataset of algorithms with optimal parameters

From the all the results, we can find out that:

Datasets	BH t-SNE	UMAP	LargeVis
Iris(N=150,M=4)	2.89	1.37	408.15
Wine(N=178,M=13)	5.55	1.43	414.15
Oliv(N=400,M=4096)	18.65	4.4	427.23
BCW(N=569,M=30)	23.15	5.39	425.95
Digits(N=1797,M=64)	49.01	9.23	535.71
LFW(N=13233,M=5749)	1181.07	98.18	1625.84

Table 6.2: Time consumption for each dataset of algorithms with optimal parameters

1. The number of dimension does not really matters the running time on these datasets for these three algorithms.

2. The AUC score of BH t-SNE and Largevis with optimal parameters are almost the same with a difference of 2 percent maximum. However, the AUC score of UMAP always has less 7 to 8 percent.

3. Considering the time complexity for BH t-SNE, LargeVis and UMAP is $O(N \log N)$[6], $O(MN)$[16](M equal to the number of negative sampling), $O(N^{1.14})$[21] separately as well as the performance of the experiment. we can draw the conclusion that when it is needed to have a fast result or better understanding of global structure of HD data, the UMAP is the best choice. If the result need to be more accurate and more attention to details of the structure in HD, when the instances of the dataset is less than 10^5 level, BH t-SNE will save more time. When it is above, LargeVis is the better one with less time consumption[2].

Chapter 7

Conclusion

This manuscript studies the three most advanced data dimensionality reduction algorithms based on random neighbor embedding methods, namely BH t-SNE, UMAP and LargeVis. This manuscript first derives three algorithms from the perspective of theory and experiment. The DR quality standard is also used to measure the performance of the three algorithms on different data sets. Then, check the more important parameters of each algorithm, and similar parameters of different algorithms, and explore the possible reasons that lead to their different results. Finally, the grid search algorithm is used to measure the best parameters of the three DR algorithms, and the similarities and differences of the algorithms running under the best conditions are compared. Experiments on actual data sets show that UMAP has better running time and retention of global structure, while LargeVis has better quality of local visualization results. Although BH t-SNE focus almost only on the local structure, the best parameter performance of its AUC value is at the same level as the other two algorithms. In the future, inspired by the neural network structure of the autoencoder, its middle layers will be compared horizontally, and the comparison dimension will be expanded from 2 dimensions to multiple dimensions.

Bibliography

[1] J. A. Lee and M. Verleysen. Quality assessment of dimensionality reduction: Rank-based criteria. Neurocomputing, 72(7):1431–1443, 2009.

[2] Jian Tang, Jingzhou Liu, Ming Zhang, Qiaozhu Mei. Visualizing Large-scale and High-dimensional Data. 2016, the web conference.

[3] Freund Y, Dasgupta S, Kabra M, et al. Learning the structure of manifolds using random projections[C]. Advances in Neural Information Processing Systems. 2007: 473-480.

[4] Goldberg Y, Levy O. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method[J]. arXiv preprint arXiv:1402.3722, 2014.

[5] C de Bodt, D Mulders, M Verleysen, JA Lee Proc. Extensive assessment of Barnes-Hut t-SNE . ESANN, 135-140, 2018.

[6] L. Van Der Maaten. Accelerating t-SNE using tree-based algorithms. Journal of Machine Learning Research, 15(1):3221–3245, 2014.

[7] J. A. Lee and M. Verleysen. Nonlinear dimensionality reduction. Springer Science Business Media, 2007.

[8] J. A. Lee, E. Renard, G. Bernard, P. Dupont, and M. Verleysen. Type 1 and 2 mixtures of Kullback-Leibler divergences as cost functions in dimensionality reduction based on similarity preservation. Neurocomputing, 112:92–108, 2013.

[9] Laurens van der Maaten, Geoffrey Hinton. Visualizing Data using t-SNE. 9(Nov):2579–2605, 2008.

[10] Rehm F, Klawonn F, Kruse R. MDS polar: a new approach for dimension reduction to visualize high dimensional data[C]//International Symposium on Intelligent Data Analysis. Springer, Berlin, Heidelberg, 2005: 316-327.

[11] G. Hinton and S. Roweis. Stochastic neighbor embedding. In NIPS, volume 15, pages 833–840, 2002.

[12] Balasubramanian M, Schwartz E L, Tenenbaum J B, et al. The isomap algorithm and topological stability[J]. Science, 2002, 295(5552): 7-7.

[13] McInnes L, Healy J, Melville J. Umap: Uniform manifold approximation and projection for dimension reduction[J]. arXiv preprint arXiv:1802.03426, 2018.

[14] J. A. Lee, E. Renard, G. Bernard, P. Dupont, and M. Verleysen. Type 1 and 2 mixtures of Kullback-Leibler divergences as cost functions in dimensionality reduction based on similarity preservation. Neurocomputing, 112:92–108, 2013.

[15] Belkin M, Niyogi P. Laplacian eigenmaps and spectral techniques for embedding and clustering[C]//Advances in neural information processing systems. 2002: 585-591.

[16] Tang J, Qu M, Wang M, et al. Line: Large-scale information network embedding[C]//Proceedings of the 24th international conference on world wide web. 2015: 1067-1077.

[17] Mattie H, Onnela J P. Generalizations of Edge Overlap to Weighted and Directed Networks[J]. arXiv preprint arXiv:1712.07110, 2017.

[18] Dasgupta S, Freund Y. Random projection trees for vector quantization[J]. IEEE Transactions on Information Theory, 2009, 55(7): 3229-3242.

[19] Pascanu R, Mikolov T, Bengio Y. On the difficulty of training recurrent neural networks[C]//International conference on machine learning. 2013: 1310-1318.

[20] Van Der Maaten L. Barnes-hut-sne[J]. arXiv preprint arXiv:1301.3342, 2013.

[21] Becht E, Dutertre C A, Kwok I W H, et al. Evaluation of UMAP as an alternative to t-SNE for single-cell data[J]. BioRxiv, 2018: 298430.

[22] Duque A F, Morin S, Wolf G, et al. Extendable and invertible manifold learning with geometry regularized autoencoders[J]. arXiv preprint arXiv:2007.07142, 2020.

[23] Hinton G E, Roweis S T. Stochastic neighbor embedding[C]//Advances in neural information processing systems. 2003: 857-864.

[24] Mohar B, Alavi Y, Chartrand G, et al. The Laplacian spectrum of graphs[J]. Graph theory, combinatorics, and applications, 1991, 2(871-898): 12.

[25] O. L. Mangasarian and W. H. Wolberg: "Cancer diagnosis via linear programming", SIAM News, Volume 23, Number 5, September 1990, pp 1 18.

[26] Gary B. Huang, Marwan Mattar, Honglak Lee, and Erik Learned-Miller. Learning to Align from Scratch. Advances in Neural Information Processing Systems (NIPS), 2012.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**